CodeWarrior®

PowerPlant Reference

Because of last-minute changes to CodeWarrior, some of the information in this manual may be inaccurate. Please read the Release Notes on the CodeWarrior CD for the latest up-to-date information.

Revised: 980830 cjm

Metrowerks CodeWarrior copyright ©1993–1998 by Metrowerks Inc. and its licensors. All rights reserved.

Documentation stored on the compact disk(s) may be printed by licensee for personal use. Except for the foregoing, no part of this documentation may be reproduced or transmitted in any form by any means, electronic or mechanical, including photocopying, recording, or any information storage and retrieval system, without permission in writing from Metrowerks Inc.

Metrowerks, the Metrowerks logo, CodeWarrior, and Software at Work are registered trademarks of Metrowerks Inc. PowerPlant and PowerPlant Constructor are trademarks of Metrowerks Inc.

All other trademarks and registered trademarks are the property of their respective owners.

ALL SOFTWARE AND DOCUMENTATION ON THE COMPACT DISK(S) ARE SUBJECT TO THE LICENSE AGREEMENT IN THE CD BOOKLET.

How to Contact Metrowerks:

U.S.A. and international	Metrowerks Corp. 9801 Metric, Suite #100 Austin, TX 78758 U.S.A.
Canada	Metrowerks Inc. 1500 du College, Suite 300 Ville St-Laurent, QC Canada H4L 5G6
Mail order	Voice: (800) 377–5416 Fax: (512) 873–4901
World Wide Web	`http://www.metrowerks.com`
Registration information	`register@metrowerks.com`
Technical support	`support@metrowerks.com`
Sales, marketing, & licensing	`sales@metrowerks.com`
CompuServe	goto `Metrowerks`

Table of Contents

LAction 59

LAction() . . . 60
~LAction() . . . 60
CanRedo() . . . 61
CanUndo() . . . 61
Finalize() . . . 61
GetDescription() . . . 62
IsDone() . . . 62
IsPostable() . . . 62
Redo() . . . 63
RedoSelf() . . . 63
Undo() . . . 63
UndoSelf() . . . 64
operator=() . . . 64
mStringResID . . . 64
mStringIndex . . . 65
mIsDone . . . 65

LActiveScroller 67

LActiveScroller() . . . 68
~LActiveScroller() . . . 69
ActiveThumbScroll() . . . 69
AdaptToNewThumbValue() . . . 70
AdjustScrollBars() . . . 70
AssignThumbProcs() . . . 70
CalcValueFromPoint() . . . 71
EndThumbTracking() . . . 71
HandleThumbScroll() . . . 72
IsTrackingThumb() . . . 72
ListenToMessage() . . . 73
StartThumbTracking() . . . 73
mThumbControl . . . 73
mTrackBarUnits . . . 74
mTrackBarPin . . . 74
mTrackBarSize . . . 74
mOriginalValue . . . 74
mTrackRect . . . 74
mValueSlop . . . 75
mVertThumbAction . . . 75
mHorizThumbAction . . . 75

LApplication 77

LApplication() . . . 78
~LApplication() . . . 78
CountSubModels() . . . 78
DoQuit() . . . 79
FindCommandStatus() . . . 79
GetPositionOfSubModel() . . . 80
GetState() . . . 81
GetSubModelByName() . . . 81
GetSubModelByPosition() . . . 82
HandleAppleEvent() . . . 83
Initialize() . . . 84
MakeMenuBar() . . . 84
MakeModelDirector() . . . 84
MakeSelfSpecifier() . . . 85
ObeyCommand() . . . 85
ProcessNextEvent() . . . 86
Run() . . . 86
SendAEQuit() . . . 87
SetSleepTime() . . . 87
ShowAboutBox() . . . 87
StartUp() . . . 88
mState . . . 88
mSleepTime . . . 88

LArray 91

LArray() . . . 93
~LArray() . . . 94
AddItem() . . . 94
AdjustAllocation() . . . 94
AdjustStorage() . . . 95
AssignItemsAt() . . . 96
AttachIterator() . . . 97
BinarySearch() . . . 97
BinarySearchByKey() . . . 98
CopyArray() . . . 98
DestroyArray() . . . 99
DetachIterator() . . . 99
FetchIndexOf() . . . 100
FetchIndexOfKey() . . . 100

FetchInsertIndexOf() 100
FetchInsertIndexOfKey() 101
FetchItemAt() 101
GetComparator() 102
GetCount() 102
GetItemPtr() 103
GetItemSize() 103
GetItemsHandle() 104
GrabItemRangeSize() 104
GrabItemSize() 105
InitArray() 105
InsertItemsAt() 106
InternalAdjustAllocation() 107
InternalCopyItem() 107
InternalMoveItem() 108
InternalSwapItems() 108
InvalidateSort() 109
IsKeptSorted() 109
IsSorted() 109
ItemsInserted() 110
ItemsRemoved() 110
LinearSearch() 111
LinearSearchByKey() 111
Lock() 112
MoveItem() 112
PeekItem() 113
PokeItem() 113
Remove() 114
RemoveItemsAt() 114
SetComparator() 115
SetKeepSorted() 115
ShiftItems() 116
Sort() 116
StoreNewItems() 117
SwapItems() 117
Unlock() 118
ValidIndex() 118
operator=() 119
mComparator 119
mDataAllocated 119

mDataStored 120
mIsSorted 120
mItemCount 120
mItemsH 120
mItemSize 121
mIteratorListHead 121
mKeepSorted 121
mLockCount 121
mOwnsComparator 121

LArrayIterator 123

LArrayIterator() 124
~LArrayIterator() 124
ArrayDied() 124
CalcNextIndex() 125
CalcPreviousIndex() 125
Current() 126
GetCurrentIndex() 126
GetNextIterator() 127
ItemsInserted() 127
ItemsRemoved() 128
Next() 128
Previous() 129
PtrToCurrent() 129
PtrToNext() 130
PtrToPrevious() 130
ResetTo() 131
SetNextIterator() 131
mNextIterator 132
mArray 132
mCurrIndex 132
mNextIndex 133

LAttachable 135

LAttachable() 135
~LAttachable() 136
AddAttachment() 136
ExecuteAttachments() 137
GetDefaultAttachable() 137
RemoveAllAttachments() 138
RemoveAttachment() 138

SetDefaultAttachable() 138
sDefaultAttachable 139
mAttachments 139

LAttachment 141

LAttachment() 142
~LAttachment() 142
Execute() 143
ExecuteSelf() 143
GetExecuteHost() 144
GetMessage() 144
GetOwnerHost() 145
SetExecuteHost() 145
SetMessage() 145
SetOwnerHost() 146
mOwnerHost 146
mMessage 146
mExecuteHost 146

LBeepAttachment 149

LBeepAttachment() 149
ExecuteSelf() 150

LBorderAttachment 151

LBorderAttachment() 151
ExecuteSelf() 152
mPenState 152
mForeColor 152
mBackColor 153

LBroadcaster 155

LBroadcaster() 156
~LBroadcaster() 156
AddListener() 156
BroadcastMessage() 157
IsBroadcasting() 158
RemoveListener() 158
StartBroadcasting() 158
StopBroadcasting() 159
mListeners 159
mIsBroadcasting 159

LButton 161
LButton() . . . 162
DrawGraphic() . . . 163
DrawSelf() . . . 163
HotSpotAction() . . . 163
HotSpotResult() . . . 163
PointIsInFrame() . . . 164
SetGraphics() . . . 164
SetGraphicsType() . . . 164
mGraphicsType . . . 165
mNormalID . . . 165
mPushedID . . . 165

LCaption 167
LCaption() . . . 168
~LCaption() . . . 168
DrawSelf() . . . 169
GetDescriptor() . . . 169
GetTextTraitsID() . . . 169
GetValue() . . . 170
SetDescriptor() . . . 170
SetTextTraitsID() . . . 170
SetValue() . . . 171
mText . . . 171
mTxtrID . . . 171

LCicnButton 173
LCicnButton() . . . 173
~LCicnButton() . . . 174
DrawSelf() . . . 174
HotSpotAction() . . . 175
HotSpotResult() . . . 175
SetCicns() . . . 175
mNormalID . . . 175
mPushedID . . . 175
mNormalCicnH . . . 176
mPushedCicnH . . . 176

LCleanupTask 177
LCleanupTask() . . . 178
~LCleanupTask() . . . 178

CleanUpAtExit() 178
DoCleanup() 178
ETSPatch() 179
sCleanupTaskList 179
sOldETSRoutine 179
sNewETSRoutine 180

LClipboard 181

LClipboard() 182
~LClipboard() 182
ExecuteSelf() 182
ExportSelf() 182
GetClipboard() 183
GetData() 183
GetDataSelf() 184
ImportSelf() 185
SetData() 185
SetDataSelf() 186
sClipboard 187
mImportPending 187
mExportPending 187

LCmdEnablerAttachment 189

LCmdEnablerAttachment() 190
ExecuteSelf() 190
mCmdToEnable 190

LColorEraseAttachment 191

LColorEraseAttachment() 191
ExecuteSelf() 192
mForeColor 192
mBackColor 192

LCommander 195

LCommander() 196
~LCommander() 197
AddSubCommander() 197
AllowSubRemoval() 197
AllowTargetSwitch() 198
AttemptQuit() 198
AttemptQuitSelf() 199
BeTarget() 199

DontBeTarget() 200
FindCommandStatus() 200
GetDefaultCommander() 201
GetLatentSub() 201
GetSuperCommander() 202
GetTarget() 202
GetTopCommander() 202
GetUpdateCommandStatus() 203
HandleKeyPress() 203
InitCommander() 203
IsOnDuty() 204
IsSyntheticCommand() 204
IsTarget() 205
ObeyCommand() 205
PostAction() 206
PostAnAction() 206
ProcessCommand() 207
ProcessCommandStatus() 208
ProcessKeyPress() 208
PutChainOnDuty() 209
PutOnDuty() 209
RemoveSubCommander() 209
RestoreTarget() 210
SetDefaultCommander() 210
SetLatentSub() 211
SetSuperCommander() 211
SetTarget() 212
SetUpdateCommandStatus() 212
SwitchTarget() 212
TakeChainOffDuty() 213
TakeOffDuty() 214
sTopCommander 214
sTarget 214
sDefaultCommander 215
sUpdateCommandStatus 215
mSuperCommander 215
mSubCommanders 215
mOnDuty 215

LCommanderPane 217

LCommanderPane() 217

~LCommanderPane() 218

LComparator 219

LComparator() 219
~LComparator() 220
Clone() 220
Compare() 220
CompareToKey() 221
GetComparator() 222
IsEqualTo() 222
IsEqualToKey() 223
sComparator 223

LControl 225

LControl() 226
~LControl() 227
BroadcastValueMessage() 227
ClickSelf() 227
FindHotSpot() 228
GetMaxValue() 228
GetMinValue() 229
GetValue() 229
GetValueMessage() 229
HotSpotAction() 230
HotSpotResult() 230
IncrementValue() 231
PointInHotSpot() 231
SetMaxValue() 232
SetMinValue() 232
SetValue() 233
SetValueMessage() 233
SimulateHotSpotClick() 233
TrackHotSpot() 234
mValueMessage 235
mValue 235
mMinValue 235
mMaxValue 235

LDataArrived 237

LDataArrived() 237
~LDataArrived() 238

GetDataBuffer() 238
GetDataSize() 239
GetRemoteAddress() 239
mDataBuffer 239
mDataSize 239
mRemoteAddress 240
mMustReleaseMemory 240

LDatabase 241

LDatabase() 241
~LDatabase() 242
CloseDatabase() 242
DecodeString() 242
EncodeString() 243
GetDatabaseState() 243
NewDatabaseCursor() 243
OpenDatabase() 244
SetAccessTimeout() 244
SetDatabaseLogin() 244
mUserName[33] 245
mPassword[33] 245
mDatabaseState 245

LDatabaseView 247

LDatabaseView() 248
~LDatabaseView() 249
BuildDelete() 249
BuildInsert() 249
BuildSelect() 250
BuildUpdate() 250
ClearSubPanes() 250
DeleteRecord() 251
DoSelect() 251
FinishCreateSelf() 251
GetDatabaseCursor() 252
InsertRecord() 252
IsInNewMode() 252
IsRecordDirty() 253
IsInNewMode() 253
ListenToMessage() 253
PopulateSubPanes() 253

SetCurrentRecord() 254
UpdateRecord() 254
WasChanged() 254
mNewMode 255
mDirtyFlag 255
mDatabaseCursor 255

LDBApplication 257

LDBApplication() 257
~LDBApplication() 258
OpenDatabase() 258
CloseDatabase() 258
GetDatabase() 259
mUserName[32] 259
mPassword[32] 259
mDatabase 259

LDBExchangeView 261

LDBExchangeView() 261
~LDBExchangeView() 262
BuildDelete() 262
BuildInsert() 262
BuildSelect() 262
BuildUpdate() 262
ClearSubPanes() 262
GetDatabaseCursor() 263
IsRecordDirty() 263
PopulateSubPanes() 263
mDBXResID 263
mAssociationResID 263

LDBPopup 265

LDBPopup() 265
~LDBPopup() 266
DoSelect() 266
FinishCreateSelf() 266
GetSelectedValue() 267
mSQLString 267
mDisplayColumn 267
mValueColumn 267
mDatabaseCursor 268

LDBTable 269
LDBTable() . . . 270
~LDBTable() . . . 270
AdjustTable() . . . 270
DoSelect() . . . 270
DrawCell() . . . 271
DrawSelf() . . . 271
FinishCreateSelf() . . . 271
GetTextTraits() . . . 272
SetTextTraits() . . . 272
mQueryString . . . 272
mTextTraitsID . . . 273
mDatabaseCursor . . . 273
LDBTableView 275
LDBTableView() . . . 276
~LDBTableView() . . . 276
AdjustTable() . . . 276
DoSelect() . . . 277
DrawCell() . . . 277
DrawSelf() . . . 277
FinishCreateSelf() . . . 278
GetTextTraits() . . . 278
SetTextTraits() . . . 278
mQueryString . . . 279
mTextTraitsID . . . 279
mDatabaseCursor . . . 279
LDefaultOutline 281
LDefaultOutline() . . . 281
DrawSelf() . . . 282
LDialogBox 283
LDialogBox() . . . 284
~LDialogBox() . . . 285
FinishCreateSelf() . . . 285
HandleKeyPress() . . . 285
ListenToMessage() . . . 285
SetCancelButton() . . . 286
SetDefaultButton() . . . 286
mDefaultButtonID . . . 287

mCancelButtonID 287
mDefaultOutline 287

LDocument **289**

LDocument() 290
~LDocument() 290
AskSaveAs() 290
AttemptClose() 291
AttemptQuitSelf() 291
Close() 292
DoAEClose() 292
DoAESave() 293
DoPrint() 293
DoRevert() 293
DoSave() 294
FindByFileSpec() 294
FindCommandStatus() 295
FindNamedDocument() 295
GetAEProperty() 295
GetDescriptor() 296
GetDocumentList() 296
HandleAppleEvent() 297
IsModified() 297
MakeCurrent() 298
MakeSelfSpecifier() 298
ObeyCommand() 298
SendAEClose() 299
SendAESaveAs() 300
SetModified() 300
UsesFileSpec() 301
sDocumentList 301
mPrintRecordH 301
mIsModified 301
mIsSpecified 302

LDragAndDrop **303**

LDragAndDrop() 303
FocusDropArea() 304
HiliteDropArea() 304
PointInDropArea() 305
mPane 305

LDragTask 307

LDragTask() . . . 307
~LDragTask() . . . 309
AddFlavors() . . . 309
AddRectDragItem() . . . 309
DoDrag() . . . 310
DropLocationIsFinderTrash() . . . 310
GetDragReference() . . . 311
GetDragRegion() . . . 311
InitDragTask() . . . 311
MakeDragRegion() . . . 312
mDragRef . . . 312
mDragRegion . . . 312
mEventRecord . . . 313

LDropArea 315

LDropArea() . . . 316
~LDropArea() . . . 316
AddDropArea() . . . 317
DoDragDrawing() . . . 317
DoDragInput() . . . 318
DoDragReceive() . . . 319
DoDragSendData() . . . 319
DragAndDropIsPresent() . . . 320
DragIsAcceptable() . . . 320
EnterDropArea() . . . 321
FindDropArea() . . . 321
FocusDropArea() . . . 322
HandleDragDrawing() . . . 322
HandleDragInput() . . . 323
HandleDragReceive() . . . 324
HandleDragSendData() . . . 325
HandleDragTracking() . . . 325
HiliteDropArea() . . . 326
InTrackingWindow() . . . 326
InsideDropArea() . . . 327
InstallHandlers() . . . 327
ItemIsAcceptable() . . . 328
LeaveDropArea() . . . 328
PointInDropArea() . . . 329

ReceiveDragItem() 329
RemoveDropArea() 330
UnhiliteDropArea() 331
mDragWindow 331
mCanAcceptCurrentDrag 331
mIsHilited 332
sDragTrackingProc 332
sDragReceiveProc 332
sDropAreaList 332
sCurrentDropArea 333
sDragHasLeftSender 333

LdtFApplication 335

LdtFApplication() 335
~LdtFApplication() 335
OpenDatabase() 336
GetFileWithPrompt() 336

LEditField 337

LEditField() 338
~LEditField() 339
AdjustCursorSelf() 340
AdjustTextWidth() 340
AlignTextEditRects() 340
BeTarget() 341
ClickSelf() 341
DisableSelf() 341
DontBeTarget() 341
DrawBox() 341
DrawSelf() 342
EnableSelf() 342
FindCommandStatus() 342
FocusDraw() 342
GetDescriptor() 342
GetMacTEH() 342
GetValue() 343
HandleKeyPress() 343
HideSelf() 343
InitEditField() 343
MoveBy() 344
ObeyCommand() 344

ResizeFrameBy() 344
RestorePlace() 344
SavePlace() 344
SelectAll() 344
SetDescriptor() 345
SetKeyFilter() 345
SetMaxChars() 345
SetTextTraitsID() 346
SetValue() 346
SpendTime() 346
TooManyCharacters() 346
UseWordWrap() 347
UserChangedText() 347
mTextEditH 348
mKeyFilter 348
mTypingAction 348
mMaxChars 348
mTextTraitsID 348
mHasBox 349
mHasWordWrap 349

LEndpoint 351

LEndpoint() 352
~LEndpoint() 352
AbortThreadOperation() 352
AckSends() 353
Bind() 353
DontAckSends() 354
DontQueueSends() 354
GetLocalAddress() 354
GetState() 355
IsAckingSends() 355
IsQueuingSends() 355
QueueSends() 356
Unbind() 356
mQueueSends 356

LEraseAttachment 357

LEraseAttachment() 357
ExecuteSelf() 358

LEventDispatcher 359
LEventDispatcher() . . . 360
~LEventDispatcher() . . . 360
AdjustCursor() . . . 360
ClickMenuBar() . . . 361
DispatchEvent() . . . 361
EventActivate() . . . 362
EventAutoKey() . . . 362
EventDisk() . . . 363
EventHighLevel() . . . 363
EventKeyDown() . . . 363
EventKeyUp() . . . 364
EventMouseDown() . . . 364
EventMouseUp() . . . 365
EventOS() . . . 365
EventResume() . . . 366
EventSuspend() . . . 366
EventUpdate() . . . 366
ExecuteAttachments() . . . 367
GetCurrentEventDispatcher() . . . 368
UpdateMenus() . . . 368
UseIdleTime() . . . 368
sCurrentDispatcher . . . 369
mSaveDispatcher . . . 369
mMouseRgnH . . . 369
LEventSemaphore 371
LEventSemaphore() . . . 371
~LEventSemaphore() . . . 372
Reset() . . . 372
Signal() . . . 372
mPostCount . . . 373
LFocusBox 375
LFocusBox() . . . 375
AttachPane() . . . 376
DontRefresh() . . . 376
DrawSelf() . . . 376
GetBoxRegion() . . . 377
HideSelf() . . . 377
Refresh() . . . 377

ShowSelf() 377

LGlobalsContext 379

LGlobalsContext() 380
~LGlobalsContext() 380
GetGlobals() 380
mSavedGlobals 380

LGroupBox 383

LGroupBox() 383
~LGroupBox() 384
DrawSelf() 384
DrawText() 385
DrawBorder() 385
CalcTextBoxFrame() 386
mFrameColor 386

LGrowZone 387

LGrowZone() 389
~LGrowZone() 390
AskListenersToFree() 390
DoGrowZone() 391
GetGrowZone() 391
GiveWarning() 392
GrowZoneCallBack() 392
MemoryIsLow() 393
SpendTime() 393
UseLocalReserve() 394
sGrowZone 395
sGrowZoneUPP 395
mLocalReserve 395
mReserveSize 395
mGiveWarning 396

LGWorld 397

LGWorld() 397
~LGWorld() 398
BeginDrawing() 398
CopyImage() 399
EndDrawing() 400
GetBounds() 400
GetMacGWorld() 400

SetBounds() 401
mMacGWorld 401
mBounds 401
mSavePort 401
mSaveDevice 402

LHandleStream 403

LHandleStream() 403
~LHandleStream() 404
DetachDataHandle() 404
GetBytes() 404
GetDataHandle() 405
PutBytes() 405
SetDataHandle() 406
SetLength() 406
mDataH 407

LIconPane 409

LIconPane() 409
~LIconPane() 410
DrawSelf() 410
SetIconID() 410
mIconID 411

LInternetAddress 413

LInternetAddress() 413
~LInternetAddress() 414
Clone() 415
GetDNSAddress() 415
GetDNSDescriptor() 415
GetHostPort() 416
GetIPAddress() 416
GetIPAddress() 416
GetIPDescriptor() 417
InternalLookupAddress() 417
InternalLookupName() 418
MakeOTDNSAddress() 418
MakeOTIPAddress() 418
SetDNSAddress() 419
SetHostPort() 419
SetIPAddress() 420

mIPAddress 420
mDNSAddress 420
mHostPort 420

LInternetMapper 423

LInternetMapper() 424
~LInternetMapper() 424
AbortThreadOperation() 424
AddressToName() 425
GetLocalAddress() 425
NameToAddress() 425

LInterruptSafeList 427

LInterruptSafeList() 427
~LInterruptSafeList() 428
Append() 428
IsEmpty() 428
Remove() 429
mQueue 429
mIteratorQueue 429

LKeyScrollAttachment 431

LKeyScrollAttachment() 432
ExecuteSelf() 432
mViewToScroll 432

LLink 433

LLink() 433
~LLink() 434
GetLink() 434
SetLink() 434
mLink 434

LListBox 437

LListBox() 438
~LListBox() 439
ActivateSelf() 439
BeTarget() 440
ClickSelf() 440
DeactivateSelf() 440
DoNavigationKey() 440
DoTypeSelection() 440

DontBeTarget() 441
DrawSelf() 441
FindCommandStatus() 441
FocusDraw() 441
GetDescriptor() 441
GetDoubleClickMessage() 442
GetFocusBox() 442
GetLastSelectedCell() 442
GetMacListH() 443
GetValue() 443
HandleKeyPress() 443
HideSelf() 443
InitListBox() 443
MakeCellVisible() 444
MoveBy() 445
ObeyCommand() 445
ResizeFrameBy() 445
RestorePlace() 445
SavePlace() 446
SelectAllCells() 446
SelectOneCell() 446
SetDescriptor() 447
SetDoubleClickMessage() 447
SetValue() 447
ShowSelf() 448
UnselectAllCells() 448
mMacListH 448
mDoubleClickMessage 448
mFocusBox 448
mTextTraitsID 449
mHasGrow 449

LListener **451**

LListener() 451
~LListener() 452
AddBroadcaster() 452
IsListening() 453
ListenToMessage() 453
RemoveBroadcaster() 454
StartListening() 454
StopListening() 455

mBroadcasters 455
mIsListening 455

LLongComparator 457

LLongComparator() 457
~LLongComparator() 458
Clone() 458
Compare() 458
GetComparator() 459
IsEqualTo() 459
sLongComparator 459

LMacTCPDNSOperation 461

LMacTCPDNSOperation() 461
~LMacTCPDNSOperation() 462
GetResultProc() 462
Int_DNSCompletionProc() 462
Int_DNSCompletionProc2() 463
WaitForCompletion() 463
mOperationListMem 464
sMacTCPDNSCompletionProc 464
sMacTCPDNSPendingOperations; 464
sDNSOperationDeleteQueue 464

LMacTCPTCPSendQueue 465

LMacTCPTCPSendQueue() 465
~LMacTCPTCPSendQueue() 466
Int_InternalSend() 466
NotifyRelease() 466
mWDS 467
mMacTCPEndpoint 467
mOperation 467

LMacTCPUDPEndpoint 469

LMacTCPUDPEndpoint() 470
~LMacTCPUDPEndpoint() 470
AbortThreadOperation() 470
AckSends() 471
Bind() 471
DontAckSends() 472
GetLocalAddress() 472
GetState() 472

Int_HandleAsyncEvent() 473
Int_UDPNotifyProc() 473
IsAckingSends() 474
ReceiveFrom() 474
SendPacketData() 475
Unbind() 475
mUDPStream 476
mLocalAddress 476
mAckSends 476
mEndpointState 476
mReceiveBuffer 477
mReceiveBufferSize 477
mMessageQueue 477
mSharedPool 477
mSendQueue 477
sUDPNotifyUPP 478

LMacTCPUDPSendQueue 479

LMacTCPUDPSendQueue() 479
~LMacTCPUDPSendQueue() 480
Int_InternalSend() 480
NotifyRelease() 480
mWDS 481
mMacTCPUDPEndpoint 481
mOperation 481

LMenu 483

LMenu() 484
~LMenu() 484
CommandFromIndex() 484
FindNextCommand() 485
GetMacMenuH() 485
GetMenuID() 486
GetNextMenu() 486
IndexFromCommand() 486
InsertCommand() 487
IsInstalled() 487
IsUsed() 488
ItemIsEnabled() 488
ReadCommandNumbers() 488
RemoveCommand() 489

RemoveItem() 489
SetCommand() 490
SetInstalled() 490
SetNextMenu() 490
SetUsed() 491
SyntheticCommandFromIndex() 491
mNextMenu 492
mMacMenuH 492
mMENUid 492
mNumCommands 492
mCommandNums 493
mIsInstalled 493
mUsed 493

LMenuBar 495

LMenuBar() 495
~LMenuBar() 496
CouldBeKeyCommand() 496
FetchMenu() 496
FindCommand() 497
FindKeyCommand() 497
FindMenuItem() 498
FindNextCommand() 499
FindNextMenu() 499
GetCurrentMenuBar() 500
InstallMenu() 500
MenuCommandSelection() 501
RemoveMenu() 501
sMenuBar 501
mMenuListHead 502

LMouseTracker 503

LMouseTracker() 503
~LMouseTracker() 503
SpendTime() 504

LMovieController 505

LMovieController() 505
~LMovieController() 506
DrawSelf() 506
SpendTime() 506

mMovie 506
mMovieController 507

LMutexSemaphore 509

LMutexSemaphore() 509
~LMutexSemaphore() 510
Signal() 510
Wait() 510
mOwner 510
mNestedWaits 510

LNetMessageQueue 511

LNetMessageQueue() 511
~LNetMessageQueue() 512
SpendTime() 512
mBroadcaster 512

LODBCApplication 513

LODBCApplication() 513
~LODBCApplication() 513
OpenDatabase() 514
mDatabase 514

LOffscreenView 515

LOffscreenView() 515
~LOffscreenView() 516
Draw() 516
DrawOffscreen() 516
EstablishPort() 517
SubImageChanged() 517
mOffscreenWorld 518
mDrawingSelf 518

LOperationListMember 519

LOperationListMember() 519
~LOperationListMember() 520
mOperation 520

LOverlappingView 521

LOverlappingView() 521
~LOverlappingView() 522
FocusDraw() 522

LPaintAttachment 523

LPaintAttachment() 524
ExecuteSelf() 524
mPenState 524
mForeColor 525
mBackColor 525

LPane 527

LPane() 530
~LPane() 530
Activate() 530
ActivateSelf() 531
AdaptToNewSurroundings() 531
AdaptToSuperFrameSize() 531
AdaptToSuperScroll() 532
AdjustCursor() 532
AdjustCursorSelf() 533
ApplyForeAndBackColors() 533
CalcLocalFrameRect() 534
CalcPortFrameRect() 534
Click() 535
ClickSelf() 535
ClickTimesAreClose() 536
Contains() 536
CountPanels() 537
Deactivate() 537
DeactivateSelf() 538
Disable() 538
DisableSelf() 538
DontRefresh() 539
Draw() 539
DrawSelf() 540
Enable() 540
EnableSelf() 541
EventMouseUp() 541
FindConstPaneByID() 542
FindDeepSubPaneContaining() 542
FindPaneByID() 543
FindShallowSubPaneContaining() 543
FindSubPaneHitBy() 544

FinishCreate() 544
FinishCreateSelf() 545
FocusDraw() 545
FocusExposed() 546
GetActiveState() 546
GetClickCount() 547
GetDefaultView() 547
GetDescriptor() 547
GetEnabledState() 548
GetFrameBinding() 548
GetFrameLocation() 548
GetFrameSize() 549
GetLastPaneClicked() 549
GetLastPaneMoused() 549
GetLocalUpdateRgn() 550
GetMacPort() 550
GetPaneID() 551
GetSuperView() 551
GetUserCon() 551
GetValue() 552
GetVisibleState() 552
GlobalToPortPoint() 552
Hide() 553
HideSelf() 553
InitPane() 553
InvalPortRect() 554
InvalPortRgn() 554
IsActive() 555
IsAreaInQDSpace() 555
IsEnabled() 556
IsHitBy() 556
IsVisible() 557
LocalToPortPoint() 557
MouseEnter() 557
MouseLeave() 558
MouseWithin() 558
MoveBy() 559
PlaceInSuperFrameAt() 559
PlaceInSuperImageAt() 560
PointIsInFrame() 561

PointsAreClose() 561
PortToGlobalPoint() 562
PortToLocalPoint() 562
PrintPanel() 563
PrintPanelSelf() 564
PutInside() 564
Refresh() 565
ResizeFrameBy() 565
ResizeFrameTo() 566
RestorePlace() 566
SavePlace() 567
ScrollToPanel() 567
SetDefaultView() 568
SetDescriptor() 568
SetForeAndBackColors() 568
SetFrameBinding() 569
SetLastPaneMoused() 569
SetPaneID() 570
SetRefreshAllWhenResized() 570
SetUserCon() 571
SetValue() 571
Show() 571
ShowSelf() 572
SuperActivate() 572
SuperDeactivate() 572
SuperDisable() 573
SuperEnable() 573
SuperHide() 573
SuperPrintPanel() 574
SuperShow() 574
UpdateClickCount() 575
UpdatePort() 575
ValidPortRect() 576
ValidPortRgn() 576
sDefaultView 576
sLastPaneClicked 577
sLastPaneMoused 577
sWhenLastMouseUp 577
sWhenLastMouseDown 577
sWhereLastMouseDown 577

sClickCount 578
mPaneID 578
mFrameSize 578
mFrameLocation 578
mFrameBinding 578
mUserCon 579
mSuperView 579
mVisible 579
mActive 579
mEnabled 580
mRefreshAllWhenResized 580

LPeriodical 581

LPeriodical() 582
~LPeriodical() 582
DeleteIdlerAndRepeaterQueues() 583
DevoteTimeToIdlers() 583
DevoteTimeToRepeaters() 584
SpendTime() 584
StartIdling() 585
StartRepeating() 585
StopIdling() 586
StopRepeating() 586
sIdlerQ 586
sRepeaterQ 587

LPicture 589

LPicture() 589
DrawSelf() 590
GetPictureID() 590
InitPicture() 591
SetPictureID() 591
mPICTid 591

LPlaceHolder 593

LPlaceHolder() 593
~LPlaceHolder() 594
CountPanels() 594
InstallOccupant() 595
RemoveOccupant() 596
ScrollToPanel() 596

mOccupant 596
mOccupantSuperView 597
mOccupantPlaceH 597
mOccupantAlignment 597

LPreferencesFile 599

LPreferencesFile() 599
OpenOrCreateResourceFork() 600

LPrintout 601

LPrintout() 602
~LPrintout() 602
ApplyForeAndBackColors() 602
CountPanels() 603
CreatePrintout() 603
DoPrintJob() 603
EstablishPort() 603
FinishCreateSelf() 604
GetMacPort() 604
GetPrintJobSpecs() 604
GetPrintRecord() 605
HasAttribute() 605
InitPrintout() 605
PageToPanel() 606
PrintCopiesOfPages() 606
PrintPanel() 607
PrintPanelRange() 607
SetAttribute() 608
SetForeAndBackColors() 608
SetPrintRecord() 608
mAttributes 609
mPrintRecordH 609
mPrinterPort 609
mWindowPort 609
mHorizPanelCount 610
mVertPanelCount 610
mForeColor 610
mBackColor 610

LQueue 611

LQueue() 611

~LQueue() . . . 612
DoForEach() . . . 612
GetSize() . . . 612
IsEmpty() . . . 613
NextGet() . . . 613
NextPut() . . . 613
Remove() . . . 614
operator =() . . . 614
mFirst . . . 614
mLast . . . 614
mSize . . . 615

LRadioGroup 617

LRadioGroup() . . . 617
~LRadioGroup() . . . 618
AddRadio() . . . 618
GetCurrentRadioID() . . . 619
ListenToMessage() . . . 619
mCurrentRadio . . . 619

LReentrantMemoryPool 621

LReentrantMemoryPool() . . . 621
~LReentrantMemoryPool() . . . 622
AddPool() . . . 622
AllocFrom() . . . 622
DisposePtr() . . . 623
FreeMem() . . . 623
GetPtrSize() . . . 624
MakeFreeBlock() . . . 624
NewPtr() . . . 624
NewPtrClear() . . . 625
TotalMem() . . . 625
mMemoryPools . . . 626
mFreeBlocks; . . . 626

LScroller 627

LScroller() . . . 628
~LScroller() . . . 629
ActivateSelf() . . . 629
AdjustScrollBars() . . . 629
DeactivateSelf() . . . 630

DrawSelf() 630
ExpandSubPane() 630
FinishCreateSelf() 630
HasHorizontalScrollBar() 630
HasVerticalScrollBar() 631
HorizSBarAction() 631
HorizScroll() 632
InstallView() 632
ListenToMessage() 633
MakeScrollBars() 634
ResizeFrameBy() 634
SubImageChanged() 635
VertSBarAction() 635
VertScroll() 636
mScrollingView 636
mVerticalBar 637
mHorizontalBar 637
mScrollingViewID 637

LSemaphore 639

LSemaphore() 639
~LSemaphore() 640
BlockThread() 640
InitSemaphore() 640
Signal() 641
UnblockAll() 641
UnblockThread() 641
Wait() 642
operator =() 643
mExcessSignals0 643
mThreads 643

LSendQueue 645

LSendQueue() 646
~LSendQueue() 646
Append() 646
GetBlockingDataSize() 647
GetMaxPendingRelease() 647
Int_InternalSend() 647
Int_SendComplete() 648
InternalClearReleaseQueue() 648

IsBusy() 648
KillQueue() 649
NotifyRelease() 649
Run() 649
SetBlockingDataSize() 650
SetMaxPendingRelease() 650
WaitingDataSize() 650
mWaitingQueue 651
mPendingQueue 651
mReleaseQueue 651
mContinue 651
mDeleteData 652
mEndpointDead 652
mBusy 652
mReleaseWaiting 652
mEndpoint 652
mThread 653
mMaxPendingRelease 653
mBlockingDataSize 653

LSharable 655

LSharable() 655
~LSharable() 655
AddUser() 656
GetUseCount() 656
NoMoreUsers() 656
RemoveUser() 657
mUseCount 657

LSharableModel 659

LSharableModel() 659
~LSharableModel() 660
AddUser() 660
Finalize() 660
SuperDeleted() 661

LSharedMemoryPool 663

LSharedMemoryPool() 663
~LSharedMemoryPool() 664
AddPoolUser() 664
GetSharedPool() 664

RemovePoolUser() 665
sSharedMemoryPool 665

LSimpleThread 667

LSimpleThread() 667
Run() 668
mProc 668
mArg 668

LSingleDoc 669

LSingleDoc() 669
~LSingleDoc() 670
AllowSubRemoval() 670
GetDescriptor() 670
GetFile() 670
GetWindow() 670
MakeCurrent() 671
UsesFileSpec() 671
mWindow 671
mFile 672

LSIOUXAttachment 673

LSIOUXAttachment() 673
ExecuteSelf() 674

LStdButton 675

LStdButton() 675
HotSpotResult() 676

LStdCheckBox 677

LStdCheckBox() 677
HotSpotResult() 678

LStdControl 679

LStdControl() 680
~LStdControl() 682
AlignControlRect() 682
CalcBigValue() 683
CalcSmallValue() 683
CreateFromCNTL() 684
DisableSelf() 684
DrawSelf() 685

EnableSelf() . . . 685
FindHotSpot() . . . 685
FocusDraw() . . . 685
GetDescriptor() . . . 685
GetMacControl() . . . 686
GetTrackingControl() . . . 686
HideSelf() . . . 686
HotSpotAction() . . . 686
HotSpotResult() . . . 687
InitStdControl() . . . 687
MoveBy() . . . 688
PointInHotSpot() . . . 688
ResizeFrameBy() . . . 688
SetActionProc() . . . 688
SetDescriptor() . . . 689
SetMaxValue() . . . 689
SetMinValue() . . . 689
SetStdMinAndMax() . . . 689
SetThumbFunc() . . . 690
SetValue() . . . 690
ShowSelf() . . . 691
TrackHotSpot() . . . 691
ValueIsInStdRange() . . . 691
mMacControlH . . . 691
mThumbFunc . . . 692
mControlKind . . . 692
mTextTraitsID . . . 692
mUsingBigValues . . . 692
sTrackingControl . . . 693

LStdPopupMenu 695

LStdPopupMenu() . . . 695
~LStdPopupMenu() . . . 697
DrawSelf() . . . 697
GetMacMenuH() . . . 697
InitStdPopupMenu() . . . 697

LStdRadioButton 699

LStdRadioButton() . . . 699
HotSpotResult() . . . 701
SetValue() . . . 701

LStr255 703

LStr255() . . . 703
LStr255(const LStr255&) . . . 704
LStr255(const LString&) . . . 704
LStr255(const unsigned char*) . . . 704
LStr255(unsigned char) . . . 705
LStr255(char) . . . 705
LStr255(const char*) . . . 705
LStr255(const void*,unsigned char) . . . 705
LStr255(char**) . . . 706
LStr255(short,short) . . . 706
LStr255(long) . . . 707
LStr255(long double,signed char,short) . . . 707
LStr255(unsigned long) . . . 707
operator=() . . . 708
mString . . . 709

LString 711

LString() . . . 712
Append (*TypeParameters*) . . . 712
AppendPStr() . . . 713
Assign(*TypeParameters*) . . . 714
BeginsWith(*TypeParameters*) . . . 715
CStringLength() . . . 716
CompareBytes() . . . 717
CompareIgnoringCase() . . . 717
CompareTo() . . . 718
CopyPStr() . . . 719
EndsWith() . . . 719
Find() . . . 720
FindWithin() . . . 721
FourCharCodeToPStr() . . . 722
Insert() . . . 722
Length() . . . 723
LoadFromPtr() . . . 723
LoadFromSTRListResource() . . . 724
LoadFromStringPtr() . . . 724
PStrToFourCharCode() . . . 725
Remove() . . . 725
Replace() . . . 726

ReverseFind() 727
SetCompareFunc() 728
ToolboxCompareText() 728
operator += (*TypeParameter*) 729
operator = (*TypeParameter*) 729
operator ()(unsigned char,unsigned char) const 730
operator [ ](unsigned char) const 730
operator [ ](unsigned char) 730
operator const unsigned char*() const 731
operator long double() const 731
operator long() const 731
operator unsigned char*() 732
operator unsigned long() const 732
mStringPtr 732
mCompareFunc 733
mMaxBytes 733

LSubOverlapView **735**

LSubOverlapView() 735
~LSubOverlapView() 736
FocusDraw() 736

LTabGroup **737**

LTabGroup() 738
~LTabGroup() 738
BeTarget() 738
GetOnDutySub() 738
HandleKeyPress() 739
RotateTarget() 739

LTable **741**

LTable() 742
~LTable() 742
ActivateSelf() 743
ClickCell() 743
ClickSelf() 743
DeactivateSelf() 743
DrawCell() 743
DrawSelf() 744
EqualCell() 744
FetchCellDataIndex() 744

FetchCellHitBy() 745
FetchLocalCellFrame() 745
GetCellData() 746
GetSelectedCell() 746
GetTableSize() 747
HiliteCell() 747
InitTable() 747
InsertCols() 748
InsertRows() 749
IsValidCell() 749
RemoveCols() 750
RemoveRows() 750
SelectCell() 751
SetCellData() 751
SetCellDataSize() 752
SetColWidth() 752
SetRowHeight() 753
UnhiliteCell() 753
mRows 754
mCols 754
mRowHeight 754
mColWidth 754
mCellData 754
mSelectedCell 755

LTableArrayStorage 757

LTableArrayStorage() 757
~LTableArrayStorage() 758
FindCellData() 758
GetCellData() 758
GetStorageSize() 758
InsertCols() 758
InsertRows() 759
RemoveCols() 759
RemoveRows() 759
SetCellData() 759
mDataArray 759
mOwnsArray 759

LTableGeometry 761

LTableGeometry() 761

~LTableGeometry() 762
GetColHitBy() 762
GetColWidth() 762
GetImageCellBounds() 763
GetRowHeight() 763
GetRowHitBy() 764
GetTableDimensions() 764
InsertCols() 765
InsertRows() 765
RemoveCols() 765
RemoveRows() 766
SetColWidth() 766
SetRowHeight() 767
mTableView 767

LTableMonoGeometry 769

LTableMonoGeometry() 769
~LTableMonoGeometry() 770
GetColHitBy() 770
GetColWidth() 770
GetImageCellBounds() 770
GetRowHeight() 770
GetRowHitBy() 770
GetTableDimensions() 771
SetColWidth() 771
SetRowHeight() 771
mColWidth 771
mRowHeight 771

LTableMultiGeometry 773

LTableMultiGeometry() 774
~LTableMultiGeometry() 774
GetColHitBy() 774
GetColWidth() 774
GetImageCellBounds() 774
GetRowHeight() 775
GetRowHitBy() 775
GetTableDimensions() 775
InsertCols() 775
InsertRows() 775
RemoveCols() 775

RemoveRows() 775
SetColWidth() 775
SetRowHeight() 775
mRowHeights 776
mColWidths 776
mDefaultRowHeight 776
mDefaultColWidth 776

LTableMultiSelector 777

LTableMultiSelector() 777
~LTableMultiSelector() 778
CellIsSelected() 778
ClickSelect() 778
DragSelect() 778
GetFirstSelectedCell() 778
GetFirstSelectedRow() 778
InsertCols() 778
InsertRows() 779
RemoveCols() 779
RemoveRows() 779
SelectAllCells() 779
SelectCell() 779
SelectCellBlock() 779
UnselectAllCells() 780
UnselectCell() 780
mSelectionRgn 780
mAnchorCell 780

LTableSelector 781

LTableSelector() 781
~LTableSelector() 782
CellIsSelected() 782
ClickSelect() 782
DragSelect() 783
GetFirstSelectedCell() 783
GetFirstSelectedRow() 784
InsertCols() 784
InsertRows() 784
RemoveCols() 785
RemoveRows() 785
SelectAllCells() 786

SelectCell() 786
UnselectAllCells() 786
UnselectCell() 787
mTableView 787

LTableSingleSelector 789

LTableSingleSelector() 789
~LTableSingleSelector() 790
CellIsSelected() 790
ClickSelect() 790
DragSelect() 790
GetFirstSelectedCell() 790
GetFirstSelectedRow() 790
InsertCols() 790
InsertRows() 791
RemoveCols() 791
RemoveRows() 791
SelectAllCells() 791
SelectCell() 791
UnselectAllCells() 791
UnselectCell() 791
mSelectedCell 791

LTableStorage 793

LTableStorage() 793
~LTableStorage() 794
FindCellData() 794
GetCellData() 794
GetStorageSize() 795
InsertCols() 795
InsertRows() 796
RemoveCols() 796
RemoveRows() 797
SetCellData() 797
mTableView 798

LTableView 799

LTableView() 801
~LTableView() 801
ActivateSelf() 801
AdjustImageSize() 802

CellIsSelected() 802
CellToIndex() 802
ClickCell() 803
ClickSelect() 803
ClickSelf() 804
DeactivateSelf() 804
DrawCell() 804
DrawSelf() 805
FetchIntersectingCells() 805
FindCellData() 805
GetCellData() 806
GetCellHitBy() 807
GetColWidth() 807
GetCustomHilite() 808
GetFirstSelectedCell() 808
GetHiliteRgn() 808
GetImageCellBounds() 809
GetLocalCellRect() 810
GetNextCell() 810
GetNextSelectedCell() 811
GetPreviousCell() 811
GetPreviousSelectedCell() 812
GetRowHeight() 812
GetTableGeometry() 813
GetTableSelector() 813
GetTableSize() 813
GetTableStorage() 814
HiliteCell() 814
HiliteCellActively() 815
HiliteCellInactively() 815
HiliteSelection() 816
IndexToCell() 816
InitTable() 817
InsertCols() 817
InsertRows() 818
IsValidCell() 818
IsValidCol() 819
IsValidRow() 819
PointsAreClose() 820
RefreshCell() 820

RefreshCellRange() 821
RemoveAllCols() 821
RemoveAllRows() 821
RemoveCols() 822
RemoveRows() 822
ScrollCellIntoFrame() 823
SelectAllCells() 823
SelectCell() 824
SelectionChanged() 824
SetCellData() 824
SetColWidth() 825
SetCustomHilite() 825
SetDeferAdjustment() 826
SetRowHeight() 826
SetTableGeometry() 827
SetTableSelector() 827
SetTableStorage() 828
SetUseDragSelect() 828
UnselectAllCells() 829
UnselectCell() 829
mRows 829
mCols 829
mTableGeometry 830
mTableSelector 830
mTableStorage 830
mUseDragSelect 830
mCustomHilite 831
mDeferAdjustment 831

LTCPEndpoint 833

LTCPEndpoint() 834
~LTCPEndpoint() 834
AbortiveDisconnect() 834
AcceptIncoming() 835
AcceptRemoteDisconnect() 835
Connect() 835
Disconnect() 836
GetAmountUnread() 836
GetRemoteHostAddress() 836
Listen() 837
Receive() 837

ReceiveChar() 838
ReceiveData() 838
ReceiveDataUntilMatch() 839
ReceiveLine() 840
RejectIncoming() 840
Send() 841
SendCStr() 841
SendData() 841
SendDisconnect() 842
SendHandle() 842
SendPStr() 843
SendPtr() 843

LTEClearAction 845

LTEClearAction() 845
~LTEClearAction() 845
RedoSelf() 846

LTECutAction 847

LTECutAction() 847
~LTECutAction() 848
RedoSelf() 848

LTEPasteAction 849

LTEPasteAction() 849
~LTEPasteAction() 850
RedoSelf() 850
UndoSelf() 850
mPastedTextH 850

LTETextAction 851

LTETextAction() 851
~LTETextAction() 852
CanRedo() 852
CanUndo() 853
Redo() 853
Undo() 853
UndoSelf() 853
mTextCommander 853
mTextPane 853
mMacTEH 853
mActionCommand 854

mDeletedTextH 854
mDeletedTextLen 854
mSelStart 854
mSelEnd 854

LTETypingAction 857

LTETypingAction() 857
~LTETypingAction() 858
BackwardErase() 858
ForwardErase() 859
InputCharacter() 859
RedoSelf() 859
Reset() 859
UndoSelf() 860
mTypedTextH 860
mTypingStart 860
mTypingEnd 860

LTextButton 863

LTextButton() 864
~LTextButton() 864
DrawSelf() 864
GetDescriptor() 864
HotSpotAction() 865
HotSpotResult() 865
SetDescriptor() 865
SetValue() 865
mText 865
mTextTraitsID 865
mSelectedStyle 866

LTextEdit 867

LTextEdit() 868
~LTextEdit() 869
AdjustCursorSelf() 869
AdjustImageToText() 869
AlignTextEditRects() 869
BeTarget() 870
ClickSelf() 870
CreateTextEditStream() 870
DontBeTarget() 871

DrawSelf() . . . 871
FindCommandStatus() . . . 871
FocusDraw() . . . 871
GetMacTEH() . . . 871
GetTextHandle() . . . 872
HandleKeyPress() . . . 872
HasAttribute() . . . 872
HideSelf() . . . 872
InitTextEdit() . . . 873
MoveBy() . . . 873
ObeyCommand() . . . 873
ResizeFrameBy() . . . 873
RestorePlace() . . . 873
SavePlace() . . . 874
SaveStateForUndo() . . . 874
ScrollImageBy() . . . 875
SelectAll() . . . 875
SetTextHandle() . . . 875
SetTextPtr() . . . 875
SetTextTraitsID() . . . 876
SpendTime() . . . 876
UserChangedText() . . . 876
mTextEditH . . . 877
mTextTraitsID . . . 877
mTextAttributes . . . 877

LToggleButton 879

LToggleButton() . . . 880
DrawGraphic() . . . 881
DrawSelf() . . . 881
HotSpotAction() . . . 882
HotSpotResult() . . . 882
PointIsInFrame() . . . 882
SetGraphics() . . . 882
SetGraphicsType() . . . 883
SetValue() . . . 883
mGraphicsType . . . 883
mOnID . . . 884
mOnClickID . . . 884
mOffID . . . 884
mOffClickID . . . 884

mTransitionID 885

LUDPEndpoint 887

LUDPEndpoint() 887
~LUDPEndpoint() 888
ReceiveFrom() 888
SendPacketData() 888
. 889

LUndoer 891

LUndoer() 891
~LUndoer() 892
ExecuteSelf() 892
FindUndoStatus() 893
PostAction() 893
ToggleAction() 894
operator=() 894
mAction 895

LVariableArray 897

LVariableArray() 898
~LVariableArray() 899
AdjustAllocation() 899
AdjustStorage() 899
AssignItemsAt() 899
GetItemOffset() 899
GetItemPtr() 900
GetItemSize() 900
GetOffsetsHandle() 901
GrabItemRangeSize() 901
GrabItemSize() 901
InternalAdjustAllocation() 901
InternalCopyItem() 902
PokeItem() 902
ShiftItems() 902
Sort() 902
StoreNewItems() 902
mItemOffsetsH 902
mItemsAllocated 902

LView 905

LView() 907

~LView() . . . 908
Activate() . . . 908
AdaptToNewSurroundings() . . . 908
AdaptToSuperFrameSize() . . . 908
AddSubPane() . . . 908
AdjustCursor() . . . 909
AutoScrollImage() . . . 909
CalcPortOrigin() . . . 909
CalcRevealedRect() . . . 910
Click() . . . 911
CountPanels() . . . 911
Deactivate() . . . 911
DeleteAllSubPanes() . . . 911
Disable() . . . 911
DontRefresh() . . . 911
Draw() . . . 912
Enable() . . . 912
EstablishPort() . . . 912
ExpandSubPane() . . . 912
FindConstPaneByID() . . . 913
FindDeepSubPaneContaining() . . . 913
FindPaneByID() . . . 913
FindShallowSubPaneContaining() . . . 913
FindSubPaneHitBy() . . . 914
FinishCreate() . . . 914
FocusDraw() . . . 914
FocusExposed() . . . 914
GetDescriptorForPaneID() . . . 914
GetImageLocation() . . . 915
GetImageSize() . . . 915
GetInFocusView() . . . 916
GetLocalUpdateRgn() . . . 916
GetPortOrigin() . . . 916
GetRevealedRect() . . . 917
GetScrollPosition() . . . 917
GetScrollUnit() . . . 917
GetSubPanes() . . . 918
GetValueForPaneID() . . . 918
Hide() . . . 919
ImagePointIsInFrame() . . . 919

ImageRectIntersectsFrame() 919
ImageToLocalPoint() 920
InitView() 920
LocalToImagePoint() 921
LocalToPortPoint() 921
MoveBy() 922
OrientAllSubPanes() 922
OrientSubPane() 922
OutOfFocus() 923
PortToLocalPoint() 923
PrintPanel() 923
ReconcileFrameAndImage() 924
Refresh() 925
RemoveSubPane() 925
ResizeFrameBy() 925
ResizeImageBy() 925
ResizeImageTo() 926
RestorePlace() 926
SavePlace() 927
ScrollBits() 927
ScrollImageBy() 927
ScrollImageTo() 928
ScrollPinnedImageBy() 929
ScrollToPanel() 930
SetDescriptorForPaneID() 930
SetReconcileOverhang() 930
SetScrollUnit() 931
SetValueForPaneID() 931
Show() 932
SubImageChanged() 932
SuperActivate() 932
SuperDeactivate() 932
SuperDisable() 932
SuperEnable() 933
SuperHide() 933
SuperPrintPanel() 933
SuperShow() 933
sInFocusView 933
mImageSize 933
mImageLocation 934

mScrollUnit 934
mPortOrigin 934
mSubPanes 934
mRevealedRect 934
mUpdateRgnH 935
mReconcileOverhang 935

LWindow **937**

LWindow() 939
~LWindow() 940
Activate() 940
ActivateSelf() 940
ApplyForeAndBackColors() 940
AttemptClose() 940
CalcStandardBounds() 941
CalcStandardBoundsForScreen() 941
ClearAttribute() 942
ClickInContent() 942
ClickInDrag() 942
ClickInGoAway() 943
ClickInGrow() 943
ClickInZoom() 944
CreateWindow() 944
Deactivate() 945
DeactivateSelf() 945
DoClose() 945
DoSetBounds() 945
DoSetPosition() 946
DoSetZoom() 946
DrawSelf() 947
DrawSizeBox() 947
Enable() 947
EstablishPort() 948
ExpandSubPane() 948
FetchWindowObject() 948
FindCommandStatus() 949
FindWindowByID() 949
GetAEProperty() 949
GetAEWindowAttribute() 950
GetDescriptor() 951
GetGlobalBounds() 951

GetMacPort() 951
GetMinMaxSize() 951
GetStandardSize() 952
GlobalToPortPoint() 952
HandleAppleEvent() 952
HandleClick() 953
HasAttribute() 954
HideSelf() 954
InitWindow() 954
InvalPortRect() 955
InvalPortRgn() 955
MakeMacWindow() 955
MakeSelfSpecifier() 955
MoveWindowBy() 956
MoveWindowTo() 956
ObeyCommand() 957
PortToGlobalPoint() 957
ResizeFrameBy() 957
ResizeWindowBy() 957
ResizeWindowTo() 958
Resume() 958
Select() 959
SendAESetBounds() 959
SendAESetPosition() 960
SendAESetZoom() 960
SetAEProperty() 961
SetAttribute() 961
SetDescriptor() 962
SetForeAndBackColors() 962
SetMinMaxSize() 962
SetStandardSize() 962
Show() 963
ShowSelf() 963
Suspend() 963
UpdatePort() 963
ValidPortRect() 963
ValidPortRgn() 964
mMacWindowP 964
mMinMaxSize 964
mStandardSize 964

mUserBounds 964
mAttributes 965
mForeColor 965
mBackColor 965
mMoveOnlyUserZoom 965

LYieldAttachment 967

LYieldAttachment() 967
ExecuteSelf() 968
mQuantum 968
mNextTicks 968

StAsyncOperation 969

StAsyncOperation() 969
~StAsyncOperation() 970
AbortOperation() 970
GetThreadOperation() 970
Int_AsyncResume() 970
WaitForResult() 971
mThread 971
mResult 971
sPendingOperations 972

StCritical 973

StCritical() 973
~StCritical() 973

StCursor 975

StCursor() 975
~StCursor() 976
mRestoreID 976

StDialogHandler 977

StDialogHandler() 978
~StDialogHandler() 978
AllowSubRemoval() 978
DoDialog() 978
FindCommandStatus() 979
GetDialog() 979
ListenToMessage() 979
SetSleepTime() 979
mDialog 980

mMessage 980
mSleepTime 980

StMacTCPOperation 981

StMacTCPOperation() 981
~StMacTCPOperation() 982
AsyncRun() 982
GetCompletionProc() 983
GetParamBlock() 983
Int_TCPCompletionProc() 983
Run() 984
mTCPParamBlock 984
sMacTCPCompletionProc 984
sTCPParamBlockDeleteQueue 984

StMacTCPUDPOperation 987

StMacTCPUDPOperation() 987
~StMacTCPUDPOperation() 988
GetCompletionProc() 988
GetParamBlock() 988
Int_UDPCompletionProc() 989
Run() 989
mUDPParamBlock 989
sMacTCPUDPCompletionProc 990
sUDPParamBlockDeleteQueue 990

StMutex 991

StMutex() 991
~StMutex() 991
mMutex 992

StOpenTptOperation 993

StOpenTptOperation() 993
~StOpenTptOperation() 994
GetCookie() 994
GetEventCode() 995
GetResultCode() 995
Int_TimerProc() 995
OperationTimedOut() 996
SetEventCode() 996
WaitForCompletion() 996
WaitForResult() 997

mNotifHandler 997
mEventCode 997
mCookieTest 997
mTestCookie 998
mResultCode 998
mCookie 998
mOperationTimeout 998
sOTOpTimerUPP 999

StSetupGlobals 1001

StSetupGlobals() 1001
~StSetupGlobals() 1002
mOldGlobals 1002

TArray 1003

TArray() 1003
~TArray() 1004
AddItem() 1004
AssignItemsAt() 1005
FetchIndexOf() 1005
FetchInsertIndexOf() 1005
FetchItemAt() 1005
FetchItemPtr() 1005
InsertItemsAt() 1006
Remove() 1006
Operator[ ] 1006

UCursor 1007

GetCurrentID() 1008
InitTheCursor() 1008
InAnimatedCursor() 1008
SetArrow() 1008
SetCurrentID() 1009
SetCross() 1009
SetInAnimatedCursor() 1009
SetIBeam() 1010
SetPlus() 1010
SetTheCursor() 1010
SetWatch() 1010
sCurrentID 1011
sInAnimatedCursor 1011

UDNSCache 1013
AddToDNSCache() . . . 1013
CheckCache() . . . 1014
CreateDNSCacheElem() . . . 1014
GetAddressFromCache() . . . 1014
GetNameFromCache() . . . 1015
sOTDNSNameCache . . . 1015
sOTDNSAddressCache . . . 1015
UMacTCPSupport 1017
GetMacTCPRefNum() . . . 1017
HasMacTCP() . . . 1017
OpenMacTCPDriver() . . . 1018
sMacTCPRefNum . . . 1018
sCloseResolverTask . . . 1018
UNetworkFactory 1019
CreateInternetMapper() . . . 1019
CreateTCPEndpoint() . . . 1020
CreateUDPEndpoint() . . . 1020
HasMacTCP() . . . 1020
HasOpenTransport() . . . 1021
HasTCP() . . . 1021
UOpenTptSupport 1023
GetOTGestalt() . . . 1023
HasOpenTransport() . . . 1024
HasOpenTransportTCP() . . . 1024
OTAddressToPPAddress() . . . 1024
StartOpenTransport() . . . 1025
sOTGestaltTested . . . 1025
sCloseOpenTptTask . . . 1025
sOTGestaltResult . . . 1025
UTextDrawing 1027
DrawWithJustification() . . . 1027

LAction

Description

LAction is a PowerPlant class that helps manage the do, undo, and redo operations. It maintains the data required to restore the previous state of menu items.

Methods

The methods in this class are:

LAction()	~LAction()
CanRedo()	CanUndo()
Finalize()	GetDescription()
IsDone()	IsPostable()
Redo()	RedoSelf()
Undo()	UndoSelf()
operator=()	

Data Members

The data members in this class are:

mStringResID	mStringIndex
mIsDone	

Operation

When you implement Undo capability in your program, you will want to modify the text of the Undo menu item of the Edit Menu in your application. For example, if the last operation of your program was to clear some text from a document window, you want to change the Undo menu item to Undo Clear. If you then chose Undo Clear from the Edit Menu, you would then want to change the menu item to Redo Clear.

See Also

LUndoer

Source files

(Action Classes)

`LAction.h`

`LAction.cp`

LAction()

Purpose Create or copy LAction objects.

Access Public

Prototypes

```
LAction(ResIDT inStringResID, Int16 inStringIndex,
     Boolean inAlreadyDone );
LAction(ResIDT inStringResID, Int16 inStringIndex,
     Boolean inAlreadyDone );
LAction( const LAction &inOriginal );
```

Parameters The parameters for these constructors are:

`ResIDT`	`inStringResID`	The resource ID of the Redo menu item text (a `'STR#'` resource). The Undo text menu item must have a `STR#` resource ID that is one higher.
`Int16`	`inStringIndex`	The index number in the `STR#` resource for the Redo menu item.
`Boolean`	`inAlreadyDone`	A value indicating whether the action is already done.
`const LAction&`	`inOriginal`	A reference to an LAction object.

~LAction()

Purpose Destroy LAction objects.

Access Public

Prototype

```
~LAction();
```

Parameters None

CanRedo()

Purpose This method tells whether an action is currently undone.

Access Virtual, Public

Prototype
```
virtual Boolean CanRedo() const;
```

Parameters None

Return Returns a Boolean true if the action is undone, false otherwise.

CanUndo()

Purpose This method tells whether an action is currently done.

Access Virtual, Public

Prototype
```
virtual Boolean CanUndo() const;
```

Parameters None

Return Returns a Boolean true if the action is done, false otherwise.

Finalize()

Purpose This method is called by the framework before deleting an action posted to an Undoer. This allows you to do something with the action before it is lost. For example, you may want to preserve the action if you are implementing multiple Undo.

Access Virtual, Public

Prototype
```
virtual void Finalize();
```

Parameters None

Return None

GetDescription()

Purpose This method allows you to retrieve the strings for Undo and Redo menu items.

Access Virtual, Public

Prototype
```
virtual void GetDescription( Str255 outRedoString,
     Str255 outUndoString) const;
```

Parameters None

Return None

IsDone()

Purpose This method returns the value of mIsDone.

Access Public

Prototype
```
Boolean IsDone() const;
```

Parameters None

Return A `Boolean` that reflects the value of mIsDone.

IsPostable()

Purpose Return whether an Action is postable, meaning that it affects the Undo state. This method always returns true. Override this method to return false for actions that are not undoable.

Access Virtual, Public

Prototype
```
virtual Boolean IsPostable() const;
```

Parameters None

Return A `Boolean` indicating whether the Action affects the Undo state.

Redo()

Purpose This method is a wrapper which calls RedoSelf() if the Action can be redone. Note that the first time this function is called, "Redo" really means "Do" the Action (unless mIsDone is set to `true` when creating the Action).

Access Virtual, Public

Prototype

```
virtual void Redo();
```

Parameters None

Return None

RedoSelf()

Purpose This method is a pure virtual method that you must supply an implementation for in your class that inherits from LAction.

Access Virtual, Public

Prototype

```
virtual void RedoSelf() = 0;
```

Parameters None

Return None

Undo()

Purpose This method is a wrapper to call UndoSelf().

Access Virtual, Public

Prototype `virtual void Undo();`

Parameters None

Return None

UndoSelf()

Purpose This method is a pure virtual method that you must supply an implementation for in your class that inherits from LAction.

Access Virtual, Public

Prototype `virtual void UndoSelf() = 0;`

Parameters None

Return None

operator=()

Purpose The assignment operator copies the values of the mStringResID, mStringIndex and mIsDone data members to the assigned object.

Access Public

Prototype `LAction& operator=( const LAction &inOriginal );`

Parameters The parameter for this operator is a `const` reference to an LAction object that you wish to make a copy of.

Return Returns an LAction reference.

mStringResID

Purpose This member contains the resource ID of the `'STR#'` resource that has the text for the redo menu items.

Access Protected

Prototype `ResIDT mStringResID;`

mStringIndex

Purpose This member contains an index number into the `'STR#'` resource for the redo item related to an action.

Access Protected

Prototype `Int16 mStringIndex;`

mIsDone

Purpose This member is true if an action is done, and false if an action is undone.

Access Protected

Prototype `Boolean mIsDone;`

LActiveScroller

Overview

LActiveScroller is a PowerPlant class that is used for scrolling views. Scroller views implement all the functionality of scroll bars, including creation, resizing, moving, hiding, showing, enabling and other operations. Sometimes you will see scrolling views referred to simply as "scrollers" in the PowerPlant documentation.

The only functional difference between LActiveScroller and LScroller is that LActiveScroller supports dynamic scrolling of the view.

Methods

The methods in this class are:

LActiveScroller()	~LActiveScroller()
ActiveThumbScroll()	AdaptToNewThumbValue()
AdjustScrollBars()	AssignThumbProcs()
CalcValueFromPoint()	EndThumbTracking()
HandleThumbScroll()	IsTrackingThumb()
ListenToMessage()	StartThumbTracking()

Data Members

The data members in this class are:

mThumbControl	mTrackBarUnits
mTrackBarPin	mTrackBarSize
mOriginalValue	mTrackRect
mValueSlop	mVertThumbAction
mHorizThumbAction	

Operation

The most important feature of a scroller view is that it has only one subpane, which is a view. The subview may contain an arbitrary number of panes and views. The net effect is that the scroller view may contain any number of panes of any type.

Source files

(Pane Classes)

`LActiveScroller.h`

`LActiveScroller.cp`

Ancestors LListener

LPane

LScroller

LView

LActiveScroller()

Purpose The constructor creates objects from the passed-in parameters.

Access Public

Prototype

```
LActiveScroller();
LActiveScroller( LStream *inStream );
LActiveScroller( const LScroller &inOriginal );
LActiveScroller( const SPaneInfo &inPaneInfo,
     const SViewInfo &inViewInfo,
     Int16 inHBLeftIndent,
     Int16 inHBRightIndent,
     Int16 inVBTopIndent,
     Int16 inVBBottomIndent,
     LView *inScrollingView);
```

Parameters These constructors have the following parameters:

`const LScroller&`	`inOriginal`	A reference to the LScroller object you want to copy.
`const SPaneInfo&`	`inPaneInfo`	A reference to the SPaneInfo object that is the super view.
`const SViewInfo&`	`inViewInfo`	A reference to the SViewInfo object that contains information about the SuperView.

`Int16`	`inHBLeftIndent`	The indentation to use on the left side for the horizontal scrolling. A good initial value for this parameter is 15.
`Int16`	`inHBRightIndent`	The indentation to use on the right side for the horizontal scrolling. A good initial value for this parameter is 15.
`Int16`	`inVBTopIndent`	The indentation to use on the top for the vertical scrolling. A good initial value for this parameter is 15.
`Int16`	`inVBBottomIndent`	The indentation to use on the bottom for the vertical scrolling. A good initial value for this parameter is 15.
`LView`	`inScrollingView`	A pointer to the view that corresponds to this Scroller.
`LStream*`	`inStream`	A pointer to a stream object that contains the information to create the LScroller object.

~LActiveScroller()

Purpose The destructor destroys the object.

Access Virtual, Public

Prototype

```
virtual ~LActiveScroller();
```

ActiveThumbScroll()

Purpose This is the static callback for the scroll bars when the thumb is tracked. The LScroller implementation embeds a pointer to the scroller object in the control reference field of the control record.

Access Static, Protected

Prototype

```
static pascal void LActiveThumbScroll();
```

Parameters None

Return None

AdaptToNewThumbValue()

Purpose This method is where to do the scroll of the scrolling view based on the new value for the tracking control.

Access Virtual, Protected

Prototype

```
virtual void AdaptToNewThumbValue(
     Int32 inNewValue);
```

Parameters This method has the following parameter:

`Int32`	`inNewValue`	This is the new value to set for the scroller.

Return None

AdjustScrollBars()

Purpose This method gets called as a result of the view scrolling. We want this to function the same as the LScroller implementation, except while we're tracking. We maintain the scroll bar during the track, so there's no need to calculate the value and draw the control twice per scroll.

This method is an override of AdjustScrollBars() in the LScroller class.

AssignThumbProcs()

Purpose Create new UPP's (Universal Procedure Pointers) and assign them to the thumbs of each of the scroll bars (if they exist).

To learn more about UPP's, refer to *Inside Macintosh: PowerPC System Software*, published by Addison-Wesley.

Access Virtual, Protected

Prototype
```
virtual void AssignThumbProcs();
```

Parameters None

Return None

CalcValueFromPoint()

Purpose Given a point, calculate the value for the control. The instance varaibles used in this calculation are initialized in the StartThumbTracking() method. The returned value is adjusted for slop.

Access Virtual, Protected

Prototype
```
virtual Int32 CalcValueFromPoint(Point inPoint);
```

Parameters This method has the following parameter:

`Point`	`inPoint`	This is the point coordinates from which the control value is calculated.

Return `Int32` indicating the new value for the control.

EndThumbTracking()

Purpose Register the fact that we're done tracking. The most important thing is to `NULL` out the instance variable that keeps track of the current tracking control. When that variable is `nil` we're assumed to not be tracking.

Access Virtual, Protected

Prototype
```
virtual void EndThumbTracking();
```

Parameters None

Return None

HandleThumbScroll()

Purpose This method is where the actual thumb action is handled.

Access Virtual, Protected

Prototype

```
virtual void HandleThumbScroll(
     LStdControl *inWhichControl);
```

Parameters This method has the following parameter:

`LStdControl*`	`inWhichControl`	This is the pointer to the standard control.

Return None

Remarks We could eliminate the click loop if we could guarantee to get called if the drag didn't change the value of the control. Currently, if the value doesn't change, no notification is issued, therefore there's no way for us to reset the tracking state.

IsTrackingThumb()

Purpose Correct way to check to see if we're currently in the tracking loop.

Access Virtual, Public

Prototype

```
virtual Boolean IsTrackingThumb();
```

Parameters None

Return `Boolean` indicating whether the tracking loop is currently executing.

ListenToMessage()

Purpose In this method we want to do exactly like we do in the LScroller implementaion except when we have finished tracking. Usually the `msg_ThumbDragged` message causes the scrolling view to adjust, but since we did live tracking, we're already scrolled to the correct location. So we just ignore that particular message.

This method is an override of ListenToMessage() in the LListener class.

StartThumbTracking()

Purpose This method sets up the tracking instance variables. We also calclulate the value slop constant so that clicks in the thumb of a scroll bar with a large scroll unit will not cause unnecessary jumps in the view.

Access Virtual, Protected

Prototype

```
virtual void StartThumbTracking(
     LStdControl *inWhichControl);
```

Parameters This method has the following parameter:

`LStdControl*`	`inWhichControl`	This is the pointer to the standard control.

Return None

mThumbControl

Purpose This data member is the pointer to the thumb standard control.

Access Protected

Prototype

```
LStdControl *mThumbControl;
```

mTrackBarUnits

Purpose This data member is the value of the track bar units.

Access Protected

Prototype
```
Int32 mTrackBarUnits;
```

mTrackBarPin

Purpose This data member is the value of the track bar pin.

Access Protected

Prototype
```
Int32 mTrackBarPin;
```

mTrackBarSize

Purpose This data member is the value of the track bar size.

Access Protected

Prototype
```
Int32 mTrackBarSize;
```

mOriginalValue

Purpose This data member is the original value of the control before any changes.

Access Protected

Prototype
```
Int32 mOriginalValue;
```

mTrackRect

Purpose This data member stores the Rect value for tracking.

Access Protected

Prototype
```
Rect mTrackRect;
```

mValueSlop

Purpose This data member stores the slop factor to apply to the value.

Access Protected

Prototype
```
Int32 mValueSlop;
```

mVertThumbAction

Purpose This data member stores the vertical thumb action UPP.

Access Protected

Prototype
```
ThumbActionUPP mVertThumbAction;
```

mHorizThumbAction

Purpose This data member stores the horizontal thumb action UPP.

Access Protected

Prototype
```
ThumbActionUPP mHorizThumbAction;
```

LApplication

Overview

LApplication is a PowerPlant class that is used for encapsulating the common behaviors of a typical Mac OS application.

Methods

The methods in this class are:

LApplication()	~LApplication()
CountSubModels()	DoQuit()
FindCommandStatus()	GetPositionOfSubModel()
GetState()	GetSubModelByName()
GetSubModelByPosition()	HandleAppleEvent()
Initialize()	MakeMenuBar()
MakeModelDirector()	MakeSelfSpecifier()
ObeyCommand()	ProcessNextEvent()
Run()	SendAEQuit()
SetSleepTime()	ShowAboutBox()
StartUp()	

Data Members

The data members in this class are:

mState	mSleepTime

Operation

You create a single object of this class in your application. This object manages the execution of the application program. It handles initialization, destruction, the main event loop, application-level events using LEventDispatcher , Apple Events, updating of the menu bar, cursor adjustment, and printing of documents.

Source files

(Commander Classes)

`LApplication.h`

`LApplication.cp`

See also LAttachable

LCommander

LDocApplication

LEventDispatcher

LMenuBar

LApplication()

Purpose The constructor initializes parameters such as the default event loop sleep time, Color QuickDraw, and some PowerPlant-specific parameters.

Access Public

Prototype
```
LApplication();
```

Parameters None

~LApplication()

Purpose The destructor destroys the application object.

Access Virtual, Public

Prototype
```
virtual ~LApplication();
```

Parameters None

CountSubModels()

Purpose This method counts the number of windows or documents open in the application.

Access Virtual, Public

Prototype

```
virtual Int32 CountSubModels( DescType inModelID )
    const;
```

Parameters The parameter for this method is:

`DescType`	`inModelID`	This is a resource descriptor parameter, which you can set to `cWindow` to count the number of windows open.

Return `Int32` indicating the number of open windows (if `inModelID == cWindow`) or open documents.

DoQuit()

Purpose Quit the Application, if the conditions specified by AttemptQuit() in LCommander() are met.

Access Virtual, Public

Prototype

```
virtual void DoQuit( Int32 inSaveOption );
```

Parameters The parameter for this method is:

`Int32`	`inSaveOption`	This is a parameter passed on to AttemptQuit() in LCommander().

Return None

FindCommandStatus()

Purpose Pass back status information on whether a Command is enabled and/or marked in a Menu.

Access Virtual, Public

Prototype

```
virtual void FindCommandStatus( CommandT inCommand,
    Boolean &outEnabled,
```

```
    Boolean &outUsesMark,
    Char16 &outMark,
    Str255 outName);
```

Parameters The parameters for this method are:

`CommandT`	`inCommand`	This is the command type being passed in.
`Boolean`	`outEnabled`	This passed-out value indicates whether the command is enabled.
`Boolean`	`outUsesMark`	This passed-out value indicates whether the command has a check mark.
`Char16&`	`outMark`	The character to use as a mark.
`Str255`	`outName`	A passed-out string containing the menu command text.

Return None

GetPositionOfSubModel()

Purpose Return the position (1 means the first) of an Apple Event SubModel within an Application.

Access Virtual, Public

Prototype

```
virtual Int32 GetPositionOfSubModel(
    DescType inModelID,
    const LModelObject *inSubModel) const;
```

Parameters The parameters for this method are:

`DescType`	`inModelID`	A resource descriptor that describes the SubModel, or you can set it to `cWindow` to find the window index using `GetMacPort()`.
`const LModelObject*`	`inSubModel`	This passed-in pointer is the address of the `LModeOb-ject` to search for.

Return `Int32` containing the position of the Apple Event SubModel.

GetState()

Purpose This method returns the value of the mState data member.

Access Inline, Public

Prototype
```
EProgramState GetState() const;
```

Parameters None

Return `EProgramState` containing the value of mState.

GetSubModelByName()

Purpose Pass back a token to an Apple Event SubModel specified by name.

Access Virtual, Public

Prototype
```
virtual void GetSubModelByName( DescType inModelID,
     Str255 inName,
     AEDesc &outToken) const;
```

Parameters The parameters for this method are:

`DescType`	`inModelID`	A resource descriptor that describes the SubModel, or you can set it to `cWindow` to search for a window by name.
`Str255`	`inName`	A string pointer containing the name of the SubModel to search for.
`AEDesc&`	`outToken`	This passed-out reference is the `AEDesc` for the specified SubModel.

Return None

GetSubModelByPosition()

Purpose Get the token of an Apple Event SubModel specified by its position.

Access Virtual, Public

Prototype

```
virtual void GetSubModelByPosition(
     DescType inModelID,
     Int32 inPosition,
     AEDesc &outToken) const;
```

Parameters The parameters for this method are:

`DescType`	`inModelID`	This is a resource descriptor parameter that describes the SubModel, or you can set it to `cWindow` to search for a window by its position.

`Int32`	`inPosition`	The position of the SubModel to retrieve a token for, 1 means first position.
`AEDesc&`	`outToken`	This passed-out reference is the `AEDesc` for the specified SubModel.

Return None

HandleAppleEvent()

Purpose This method handles the Apple Events that are sent to the application.

Access Virtual, Public

Prototype

```
virtual void HandleAppleEvent(
     const AppleEvent &inAppleEvent,
     AppleEvent &outAEReply,
     AEDesc &outResult,
     long inAENumber);
```

Parameters The parameters for this method are:

`const AppleEvent&`	`inAppleEvent`	This is a reference to the AppleEvent passed-in for the application to handle.
`AppleEvent&`	`outAEReply`	This is a reference to the reply received after the AppleEvent.
`AEDesc&`	`outResult`	This passed-out reference is the `AEDesc` for the specified SubModel.
`long`	`inAENumber`	This is the passed-in Apple Event number, something like `ae_Clone` or `ae_Move`.

Initialize()

Purpose Calling this method provides a last chance to initialize the application before event processing begins.

Access Virtual, Protected

Prototype
```
virtual void Initialize();
```

Parameters None

Return None

MakeMenuBar()

Purpose Create the MenuBar object for the application. You should override this to use a class other than LMenuBar.

Access Protected

Prototype
```
virtual void MakeMenuBar();
```

Parameters None

Return None

MakeModelDirector()

Purpose Create ModelDirector (an AppleEvent handle) object for the application. You should override this if you want to use a class other than LModelDirector.

Access Virtual, Protected

Prototype
```
virtual void MakeModelDirector();
```

Parameters None

Return None

MakeSelfSpecifier()

Purpose Make an Object Specifier for the application.

Access Virtual, Protected

Prototype

```
virtual void MakeSelfSpecifier(
     AEDesc& inSuperSpecifier,
     AEDesc& outSelfSpecifier) const;
```

Parameters The parameters for this method are:

`AEDesc&`	`inSuperSpecifier`	This passed-in reference is an `AEDesc`, and is not needed in this method implementation.
`AEDesc&`	`outSelfSpecifier`	This is the passed-out `AEDesc` that contains the Object Specifier created.

Return None

ObeyCommand()

Purpose Process commands that are generated when running the application.

Access Virtual, Public

Prototype

```
virtual Boolean ObeyCommand( CommandT inCommand,
     void *ioParam);
```

Parameters The parameters for this method are:

`CommandT`	`inCommand`	This is the command type being passed in.
`void*`	`ioParam`	This is a pointer to a data block that accompanies the command.

Return A `Boolean` containing `true` if the command was handled by this call, and `false` otherwise.

ProcessNextEvent()

Purpose Retrieve and handle the next event in the application event queue. This includes letting Attachments process the event, updating menu item status, and calling Repeaters.

Access Virtual, Public

Prototype
```
virtual void ProcessNextEvent();
```

Parameters None

Return None

Run()

Purpose Run the Application by processing events until quitting. This includes doing some menu management, cursor updating, and other housekeeping.

Access Virtual, Public

Prototype
```
virtual void Run();
```

Parameters None

Return None

Remarks You should catch all exceptions in your code. If an exception occurs when calling ProcessNextEvent(), a signal will occur and the application will continue running.

SendAEQuit()

Purpose Sends a Quit AppleEvent to this application.

Access Virtual Public

Prototype
```
virtual void SendAEQuit();
```

Parameters None

Return None

SetSleepTime()

Purpose This method sets the value of the mSleepTime data member.

Access Inline, Public

Prototype
```
void SetSleepTime( Int32 inSleepTime);
```

Parameters The parameter for this method is:

`Int32`	`inSleepTime`	This is the value to set for mSleepTime.

Return None

ShowAboutBox()

Purpose Display the About Box for the application. This basic implementation just puts up an Alert Box. You should override this if you wish to display a more elaborate About Box.

Access Virtual, Public

Prototype `virtual void ShowAboutBox();`

Parameters None

Return None

StartUp()

Purpose This method is meant to provide a place to perform actions at application start up when launched without any documents. You should override this if you wish to instead perform some default action, such as creating a new, untitled document.

Access Virtual, Protected

Prototype `virtual void StartUp();`

Parameters None

Return None

mState

Purpose This data member contains a value signifying the state of the program, such as `programState_StartingUp`.

Access Protected

Prototype `EProgramState mState;`

mSleepTime

Purpose This data member contains the overridable value of the sleep time parameter for the main event loop. This member is set to a default value by the constructor.

Access Protected

Prototype `long mSleepTime;`

LArray

Description

LArray is a PowerPlant class that is used for implementing an ordered collection of fixed-size items. The first item is at index value 1. The index value 0 is used to indicate a nonexistent item.

Methods

The methods in this class are:

LArray()
~LArray()
AddItem()
AdjustAllocation()
AdjustStorage()
AssignItemsAt()
AttachIterator()
BinarySearch()
BinarySearchByKey()
CopyArray()
DestroyArray()
DetachIterator()
FetchIndexOf()
FetchIndexOfKey()
FetchInsertIndexOf()
FetchInsertIndexOfKey()
FetchItemAt()
GetComparator()
GetCount()
GetItemPtr()
GetItemSize()
GetItemsHandle()
GrabItemRangeSize()
GrabItemSize()
InitArray()
InsertItemsAt()
InternalAdjustAllocation()
InternalCopyItem()
InternalMoveItem()
InternalSwapItems()
InvalidateSort()
IsKeptSorted()
IsSorted()
ItemsInserted()
ItemsRemoved()
LinearSearch()
LinearSearchByKey()
Lock()
MoveItem()
PeekItem()
PokeItem()
Remove()

RemoveItemsAt() SetComparator()
SetKeepSorted() ShiftItems()
Sort() StoreNewItems()
SwapItems() Unlock()
ValidIndex() operator=()

Data Members

The data members in this class are:

mComparator mDataAllocated
mDataStored mIsSorted
mItemCount mItemsH
mItemSize mIteratorListHead
mKeepSorted mLockCount
mOwnsComparator

Operation

Index values are signed, 32-bit integers. When specifying an item, you pass a pointer to the item data as a parameter. The array stores a copy of the data, or returns a copy of the data to you.

You can insert, remove, get a value, assign a value, swap, move items, and get data about the array. PowerPlant defines the constants `index_First` and `index_Last` so that you can easily insert items at the beginning or end of the array.

Source files

(Array Classes)

`LArray.h`

`LArray.cp`

See Also

LArrayIterator

LComparator

LLockedArrayIterator

LArray()

Purpose The constructors create the object with the passed-in parameters.

Access Public

Prototypes

```
LArray();
LArray( const LArray &inOriginal );
```

These constructors create an array with space pre-allocated for the specified number of items of the specified size. If nItemCount is not specified, then the array will be empty after creation.

```
LArray( Uint32 inItemCount,
     Uint32 inItemSize,
     LComparator *inComparator = nil,
     Boolean inKeepSorted = false );
LArray( Uint32 inItemSize,
     LComparator *inComparator = nil,
     Boolean inKeepSorted = false );
LArray( Uint32 inItemSize,
     Handle inItemsHandle,
     LComparator *inComparator = nil,
     Boolean inIsSorted = false,
     Boolean inKeepSorted = false );
```

Parameters The parameters for this constructor are:

`Uint32`	`inItemCount`	The number of items to put in the array.
`Uint32`	`inItemSize`	The size of one item.
`Handle`	`inItemsHandle`	A handle to items for the array.
`LComparator*`	`inComparator`	The Comparator object compares array items and must be allocated in the heap via "new". The array assumes ownership of the Comparator and is responsible for deleting it. The default is `nil`.

`Boolean`	`inKeepSorted`	This specifies whether to keep the array sorted when items are inserted or assigned new values.The default is `false`.
`const LArray &`	`inOriginal`	A reference to the object to copy.

~LArray()

Purpose The destructor destroys the array.

Access Virtual, Public

Prototype

```
virtual ~LArray();
```

AddItem()

Purpose Add one item to array. If the array is not sorted, add the item to the end of the array.

Access Virtual, Public

Prototype

```
virtual ArrayIndexT AddItem( const void *inItem,
    Uint32 inItemSize );
```

Parameters This method takes the following parameters:

`const void*`	`inItem`	The item to add to the array.
`Uint32`	`inItemSize`	The size of the item in bytes.

Return Returns the index point at which the item was inserted.

Remarks For unsorted Arrays, this function is a faster version of InsertItemsAt() since it doesn't have to bother with checking/adjusting the count and insertion index.

AdjustAllocation()

Purpose This method is a wrapper call to InternalAdjustAllocation().

Access Virtual, Public

Prototype

```
virtual void AdjustAllocation(
     Uint32 inExtraItems,
     Uint32 inExtraData );
```

Parameters This method takes the following parameters:

`Uint32`	`inExtraItems`	The item to add to the array.
`Uint32`	`inExtraData`	The size of the item in bytes. This has a default parameter value of zero and is ignored.

Return None

Remarks For fixed-size items, the number of items determines the amount of data stored.

AdjustStorage()

Purpose This method is called internally when the number of bytes used by items in the array changes.

Access Virtual, Protected

Prototype

```
virtual void AdjustStorage(
     Int32 inDeltaItems,
     Int32 inDeltaData );
```

Parameters This method takes the following parameters:

`Int32`	`inDeltaItems`	The change in the number of items in the array.
`Int32`	`inDeltaData`	Ignored, since the number of items determines the amount of data stored for fixed-size item arrays.

Return None

Remarks If the current allocation is too small, this implementation sets the internal allocation size to:

```
current_alloc + max(current_alloc, delta_bytes)
```

For small adjustments (adding less bytes than what's already allocated), this doubles the allocation.

For large adjustments (adding more bytes than what's already allocated), this increases the allocation by the number of bytes added.

AssignItemsAt()

Purpose Assign the same value to items in the array starting at the specified index.

Access Virtual, Public

Prototype

```
virtual ArrayIndexT AssignItemsAt(
     Uint32 inCount,
     ArrayIndexT inAtIndex,
     const void *inValue,
     Uint32 inItemSize)
```

Parameters This method takes the following parameters:

`Uint32`	`inCount`	The number of items to make the same.
`ArrayIndexT`	`inAtIndex`	The starting index.
`const void*`	`inValue`	A pointer to the item data. The array makes and stores a copy of the item data.
`Uint32`	`inItemSize`	The size of the array item.

Return Returns index of first "assigned" item. This may be different from `inAtIndex` if the array is sorted. Returns `LArray::index_Bad` if `inAtIndex` is out of range.

Remarks Does nothing if `inAtIndex` is out of range.

AttachIterator()

Purpose Associate an iterator with an array.

Access Protected

Prototype

```
void AttachIterator(
     LArrayIterator *inIterator ) const;
```

Parameters This method takes the following parameter:

`LArrayIterator*`	`inIterator`	The iterator to associate with the array.

Return None

BinarySearch()

Purpose This method returns the index of the specified item using a binary search. It assumes that the array is sorted.

Access Protected

Prototype

```
ArrayIndexT BinarySearch(
     const void *inItem,
     Uint32 inItemSize) const;
```

Parameters This method takes the following parameters:

`const void *`	`inItem`	A pointer to the item to search for.
`Uint32`	`inItemSize`	The size of the array item.

Return The index at which the item was found. If not found, the value returned will be `index_Bad`.

BinarySearchByKey()

Purpose Return the index of the item with the specified key, using a binary search. It assumes that the array is sorted.

Access Protected

Prototype

```
ArrayIndexT BinarySearchByKey(
     const void *inKey) const;
```

Parameters This method takes the following parameter:

`const void *`	`inKey`	A pointer to the key to search for.

Return The index at which the item was found. If not found, the value returned will be `index_Bad`.

CopyArray()

Purpose Creates a deep copy by duplicating the items in the Array. However, if the items in the Array are pointers to other objects, those other objects aren't duplicated.

Access Private

Prototype

```
void CopyArray( const LArray &inOriginal );
```

Parameters This method takes the following parameter:

`const LArray&`	`inOriginal`	A reference to the original object to copy.

Return None

Remarks

- Also duplicates the Comparator if the original owns its Comparator. NOTE: If you have an array which owns a custom Comparator, you MUST override Clone() for your subclass of LComparator in order for this array copy constructor to work properly.

- Copy is always unlocked, even if original is locked. This ensures that Lock() and Unlock() calls always balance. If the copy were locked (because the original is locked), then to change the copy you would have to unlock it without ever having called Lock(). And tracking the number of times the copy is locked would be a big problem.
- Copy does NOT duplicate original's iterators, since there would be no way to access such iterators. The list of iterators is an internal implementation detail.

DestroyArray()

Purpose Destroy the internal data an array.

Access Private

Prototype

```
void DestroyArray();
```

Parameters None

Return None

DetachIterator()

Purpose Remove the association of an iterator from an array.

Access Protected

Prototype

```
void DetachIterator(
     LArrayIterator *inIterator) const;
```

Parameters This method takes the following parameter:

`LArrayIterator*`	`inIterator`	A pointer to the iterator.

Return None

FetchIndexOf()

Purpose Returns the index of the specified item within the array.

Access Virtual, Public

Prototype

```
virtual ArrayIndexT FetchIndexOf(
     const void *inItem,
     Uint32 inItemSize)
```

Parameters This method takes the following parameters:

`const void *`	`inItem`	A pointer to the item to search for.
`Uint32`	`inItemSize`	The size of the array item.

Return Returns `index_Bad` if the item is not in the array.

FetchIndexOfKey()

Purpose Return the index of the item with the specified Key value.

Access Virtual, Public

Prototype

```
virtual ArrayIndexT FetchIndexOfKey(
     const void *inKey)
```

Parameters This method takes the following parameter:

`const void *`	`inKey`	A pointer to the key to search for.

Return Returns the index of the item.

FetchInsertIndexOf()

Purpose Return the index at which the specified item would be inserted.

Access Virtual, Public

Prototype

```
virtual ArrayIndexT FetchInsertIndexOf(
     const void *inItem,
     Uint32 inItemSize)
```

Parameters This method takes the following parameters:

`const void *`	`inItem`	A pointer to the item to search for.
`Uint32`	`inItemSize`	The size of the array item.

Return Returns `index_Last` if the Array is not sorted or if the item is `nil`.

FetchInsertIndexOfKey()

Purpose Return the index at which an item with the specified Key would be inserted.

Access Virtual, Public

Prototype

```
virtual ArrayIndexT FetchInsertIndexOfKey(
     const void *inKey)
```

Parameters This method takes the following parameter:

`const void *`	`inKey`	A pointer to the key to search for.

Return Returns `index_Last` if the Array is not sorted or if the item is `nil`.

FetchItemAt()

Purpose Pass back the item at the specified index.

Access Virtual, Public

Prototype

```
virtual Boolean FetchItemAt(
     ArrayIndexT inAtIndex,
     void *outItem,
     Uint32 &ioItemSize ) const;
```

```
virtual Boolean FetchItemAt(
     ArrayIndexT inAtIndex,
     void *outItem ) const;
```

Parameters This method takes the following parameters:

`ArrayIndexT`	`inAtIndex`	The index to retrieve from.
`void*`	`outItem`	The pointer to the retrieved item. Caller must make sure that this points a buffer large enough to hold the item data.
`Uint32&`	`ioItemSize`	The size of the item.

Return Returns `true` if an item exists at `inIndex` (and sets `outItem`). Returns `false` if `inIndex` is out of range (and leaves `outItem` unchanged).

GetComparator()

Purpose This method returns the value of the mComparator data member.

Access Public

Prototype

```
LComparator* GetComparator() const;
```

Parameters None

Return A pointer to the LComparator.

GetCount()

Purpose This method returns the value of the mItemCount data member.

Access Public

Prototype

```
Uint32 GetCount() const;
```

Parameters None

Return `Uint32` indicating the value of mItemCount.

GetItemPtr()

Purpose Returns a pointer to the start of an item's data within the internal storage Handle.

Access Virtual, Public

Prototype

```
virtual void* GetItemPtr(
     ArrayIndexT inAtIndex) const;
```

Parameters This method takes the following parameter:

`ArrayIndexT`	`inAtIndex`	The index to retrieve from.

Return A pointer to the start of the item data.

Remarks WARNING: The return pointer references information inside a relocatable block. This pointer will become invalid if the Handle block moves. Call Lock() and then Unlock() where necessary.

WARNING: For sorted arrays, be careful when changing the data using the pointer. If your changes alter the sorting order, call InvalidateSort() so that the array's internal flags correctly reflect the sorting state. Then call Sort() afterwards if you still want the array to be sorted.

GetItemSize()

Purpose This method returns the value of the mItemSize data member.

Access Virtual, Public

Prototype

```
virtual Uint32 GetItemSize(ArrayIndexT inIndex)
     const;
```

Parameters This method takes the following parameter:

`ArrayIndexT`	`inIndex`	The index to retrieve the size of.

Return The value of mItemSize in a `Uint32`.

GetItemsHandle()

Purpose Return Handle used to store data for array items.

Access Public

Prototype

```
Handle GetItemsHandle() const;
```

Parameters None

Return The handle to the array items.

GrabItemRangeSize()

Purpose This method returns the total size in bytes of the items in the array between (inclusive) the specified start and end indices.

Access Inline, Virtual, Protected

Prototype

```
virtual Uint32 GrabItemRangeSize(
     ArrayIndexT inStartIndex,
     ArrayIndexT inEndIndex) const;
```

Parameters This method takes the following parameters:

`ArrayIndexT`	`inStartIndex`	The starting index.
`ArrayIndexT`	`inEndIndex`	The ending index.

Return A size in bytes stored in a `Uint32`.

GrabItemSize()

Purpose This data member returns the value of the mItemSize data member.

Access Virtual, Protected

Prototype

```
virtual Uint32 GrabItemSize(
     ArrayIndexT inIndex ) const;
```

Parameters This method takes the following parameter:

`ArrayIndexT`	`inIndex`	The index (unused).

Return `Uint32` containing the value of the mItemSize member.

InitArray()

Purpose This is an internal method which initializes the data members for an array.

Access Private

Prototype

```
void InitArray(
     Uint32 inItemSize,
     LComparator *inComparator,
     Boolean inIsSorted,
     Boolean inKeepSorted);
```

Parameters This method takes the following parameters:

`Uint32`	`inItemSize`	The size of the items for the array.
`LComparator*`	`inComparator`	The pointer to the comparator object.

`Boolean`	`inIsSorted`	A value indicating whether the array is sorted or not.
`Boolean`	`inKeepSorted`	A value indicating whether to keep the array sorted or not.

Return None

InsertItemsAt()

Purpose Insert items at the specified position in an array.

Access Virtual, Public

Prototype

```
virtual ArrayIndexT InsertItemsAt(
    Uint32 inCount,
    ArrayIndexT inAtIndex,
    const void *inItem,
    Uint32 inItemSize);
```

Parameters This method takes the following parameters:

`Uint32`	`inCount`	The number of items to insert.
`ArrayIndexT`	`inAtIndex`	The index to begin inserting at.
`const void*`	`inItem`	A pointer to the item to insert.
`Uint32`	`inItemSize`	The size of the item in bytes.

Return The index at which items were inserted. This can differ from the input value of inAtIndex as described in the Remarks.

Remarks All items are set to the same value, as specified by `inItem`. `inItem` may be `nil`, in which case the data for the inserted items is unspecified (but space is allocated).

`inAtIndex` is adjusted if necessary:

- > to sorted position if array is kept sorted

- > to after last item if `inAtIndex` is too big
- > to 1 if `inAtIndex` is too small

InternalAdjustAllocation()

Purpose Called internally to change the size of the storage used.

Access Virtual, Protected

Prototype

```
virtual void InternalAdjustAllocation(
     Uint32 inItemAllocation,
     Uint32 inDataAllocation );
```

Parameters This method takes the following parameter:

`Uint32`	`inItemAllocation`	The the number of items to allocate space for.
`Uint32`	`inDataAllocation`	The the amount of data to allocate space for.

Remarks Fixed-size item Array only stores data, so inItemAllocation is ignored.

InternalCopyItem()

Purpose Set value of destination item to that of the source item.

Access Virtual, Protected

Prototype

```
virtual void InternalCopyItem(
     ArrayIndexT inSourceIndex,
     ArrayIndexT inDestIndex);
```

Parameters This method takes the following parameters:

`ArrayIndexT`	`inSourceIndex`	The source index.
`ArrayIndexT`	`inDestIndex`	The destination index.

Return None

InternalMoveItem()

Purpose Move an item from one position to another in an array. The net result is the same as removing the item and inserting at a new position.

Access Virtual, Protected

Prototype

```
virtual void InternalMoveItem(
     ArrayIndexT inFromIndex,
     ArrayIndexT inToIndex,
     void *inBuffer);
```

Parameters This method takes the following parameters:

`void*`	`inBuffer`	A pointer to the data buffer.
`ArrayIndexT`	`inSourceIndex`	The source index.
`ArrayIndexT`	`inDestIndex`	The destination index.

Return None

InternalSwapItems()

Purpose Swap the values of the Items at the specified indices. This is an internal method for this class.

Access Virtual, Protected

Prototype

```
virtual void InternalSwapItems(
     ArrayIndexT inIndexA,
     ArrayIndexT inIndexB,
     void *inBuffer);
```

Parameters This method takes the following parameters:

`void*`	`inBuffer`	A pointer to the data buffer.
`ArrayIndexT`	`inIndexA`	The first index.
`ArrayIndexT`	`inIndexB`	The second index.

Return None

InvalidateSort()

Purpose This method sets the mIsSorted data member to `false`.

Access Public

Prototype
```
void InvalidateSort();
```

Parameters None

Return None

IsKeptSorted()

Purpose This method returns the value of the mKeepSorted data member.

Access Inline, Public

Prototype
```
Boolean IsKeptSorted() const;
```

Parameters None

Return `Boolean` `true` or `false` indicating the value of mKeepSorted.

IsSorted()

Purpose This method returns the value of the mIsSorted data member.

Access Inline, Public

Prototype `Boolean IsSorted() const;`

Parameters None

Return `Boolean` `true` or `false` indicating the value of mIsSorted.

ItemsInserted()

Purpose Notify Iterators associated with an array that items have been inserted.

Access Protected

Prototype
```
void ItemsInserted(
     Uint32 inCount,
     ArrayIndexT inAtIndex);
```

Parameters This method takes the following parameters:

`Uint32`	`inCount`	The number of items that have been inserted.
`ArrayIndexT`	`inAtIndex`	The index of the insertion.

Return None

ItemsRemoved()

Purpose Notify Iterators associated with an Array that items have been removed

Access Protected

Prototype
```
void ItemsRemoved(
     Uint32 inCount,
     ArrayIndexT inAtIndex);
```

Parameters This method takes the following parameters:

`Uint32`	`inCount`	The number of items that have been removed.
`ArrayIndexT`	`inAtIndex`	The index of the removal.

Return None

LinearSearch()

Purpose Return the index of the specified item, searching linearly from the start of the array.

Access Protected

Prototype

```
ArrayIndexT LinearSearch(
     const void *inItem,
     Uint32 inItemSize) const;
```

Parameters This method takes the following parameters:

`Uint32`	`inItemSize`	The size of the item to search for.
`const void*`	`inItem`	The pointer to the item to search for.

Return The index at which the item is located, `index_Bad` if not found.

LinearSearchByKey()

Purpose Return the index of the item with the specified key, searching linearly from the start of the array.

Access Protected

Prototype

```
ArrayIndexT LinearSearchByKey(
     const void *inKey) const;
```

Parameters This method takes the following parameter:

`const void*`	`inKey`	A pointer to the key to search for.

Return The index at which the item is located, `index_Bad` if not found.

Lock()

Purpose Lock the Handle that stores the data for the items in the array.

Access Public

Prototype

```
void Lock() const;
```

Parameters None

Return None

Remarks This class maintains a lock count, so each call to Lock() should be balanced by a corresponding call to Unlock().

MoveItem()

Purpose Move an item from one position to another in an array. The net result is the same as removing the item and inserting at a new position.

Access Virtual, Public

Prototype

```
virtual void MoveItem(
     ArrayIndexT inFromIndex,
     ArrayIndexT inToIndex );
```

Parameters This method takes the following parameters:

`ArrayIndexT`	`inFromIndex`	The source index.
`ArrayIndexT`	`inToIndex`	The destination index.

Return None

Remarks Does nothing if either index is out of range or if the array is kept sorted (since moving could invalidate the sort).

PeekItem()

Purpose Pass back the data for the item at the specified index. This is used internally to read item data.

Access Virtual, Protected

Prototype

```
virtual void PeekItem(
     ArrayIndexT inAtIndex,
     void *outItem ) const;
```

Parameters This method takes the following parameters:

`ArrayIndexT`	`inAtIndex`	The array index to peek at.
`void *`	`outItem`	The pointer to the item to return data for.

Return None

PokeItem()

Purpose Store data for the item at the specified index.

Access Virtual, Protected

Prototype

```
virtual void PokeItem(
     ArrayIndexT inAtIndex,
     const void *inItem,
     Uint32 inItemSize );
```

Parameters This method takes the following parameters:

`ArrayIndexT`	`inAtIndex`	The array index to poke at.
`const void*`	`outItem`	The pointer to the item to store data for.
`Uint32`	`inItemSize`	The size of the data block to store

Return None

Remove()

Purpose Remove an item from an array.

Access Virtual, Public

Prototype

```
virtual void Remove(
     const void *inItem,
     Uint32 inItemSize);
```

Parameters This method takes the following parameters:

`const void *`	`outItem`	The pointer to the item to remove data for.
`Uint32`	`inItemSize`	The size of the data block to remove.

Return None

RemoveItemsAt()

Purpose Remove items from an array starting at a specified position.

Access Virtual, Public

Prototype

```
virtual void RemoveItemsAt(
     Uint32 inCount,
     ArrayIndexT inAtIndex);
```

Parameters This method takes the following parameters:

`ArrayIndexT`	`inAtIndex`	The index to start removing at.
`Uint32`	`inItemSize`	The size of the data block to remove.

Return None

Remarks Does nothing if inAtIndex is out of range. Checks if `inCount` would remove items past the end of the array, and adjusts it accordingly to remove the items from `inAtIndex` to the end of the array. That means you can pass a large number to remove the items from `inAtIndex` to the end of the array.

SetComparator()

Purpose Specify the comparator for items in an array.

Access Public

Prototype

```
void SetComparator(
     LComparator *inComparator,
     Boolean inTakeOwnership);
```

Parameters This method takes the following parameters:

`LComparator*`	`inComparator`	The comparator to use.
`Boolean`	`inTakeOwnership`	Indicates the value to set for mOwnsComparator.

Return None

SetKeepSorted()

Purpose Specify whether to keep an array sorted when items change.

Access Public

Prototype `void SetKeepSorted( Boolean inKeepSorted );`

Parameters This method takes the following parameter:

`Boolean`	`inKeepSorted`	Whether to keep the array sorted or not.

Return None

ShiftItems()

Purpose is an internal method that moves items within the Handle used for internal storage. It moves items in the range `inStartIndex` to `inEndIndex` (inclusive).

Access Virtual, Protected

Prototype

```
virtual void ShiftItems(
     ArrayIndexT inStartIndex,
     ArrayIndexT inEndIndex,
     Int32 inIndexShift,
     Int32 inDataShift);
```

Parameters This method takes the following parameters:

`ArrayIndexT`	`inStartIndex`	The index to start shifting at.
`ArrayIndexT`	`inEndIndex`	The index to end shifting at.
`Int32`	`inIndexShift`	The amount to shift.
`Int32`	`inDataShift`	This is unused.

Return None

Sort()

Purpose Sort items in the array.

Access Virtual, Public

Prototype `virtual void Sort();`

Parameters None

Return None

StoreNewItems()

Purpose This is an internal method that stores values within the internal storage Handle. Items all have the same value, and space must already have been allocated for them.

Access Virtual, Protected

Prototype

```
virtual void StoreNewItems(
     Uint32 inCount,
     ArrayIndexT inAtIndex,
     const void *inItem,
     Uint32 inItemSize);
```

Parameters This method takes the following parameters:

`Uint32`	`inCount`	The number of items.
`ArrayIndexT`	`inAtIndex`	The index to start storing at.
`const void*`	`inItem`	The item to store.
`Uint32`	`inItemSize`	The size of the item to store.

Return None

SwapItems()

Purpose Swap the values of the Items at the specified indices. This method does nothing if either index is out of range or if array is kept sorted (since swapping could invalidate the sort).

Access Virtual, Public

Prototype

```
virtual void SwapItems(
     ArrayIndexT inIndexA,
     ArrayIndexT inIndexB);
```

Parameters This method takes the following parameters:

`ArrayIndexT`	`inIndexA`	The first index.
`ArrayIndexT`	`inIndexB`	The second index.

Return None

Unlock()

Purpose Unlock the Handle that stores the data for the items in the array. This class maintains a lock count, so each call to Lock() should be balanced by a corresponding call to `Unlock()`.

Access Public

Prototype

```
void Unlock() const;
```

Parameters None

Return None

ValidIndex()

Purpose Indicate whether an index is valid (between 1 and the number of items) for the array.

Access Public

Prototype

```
Boolean ValidIndex(
     ArrayIndexT &ioIndex ) const;
```

Parameters This method takes the following parameter:

`ArrayIndexT&`	`ioIndex`	The reference to the index to check. If `ioIndex` is the special flag `index_Last`, the index's value is changed to the actual index of the last item.

Return `Boolean` indicating whether the index is a valid one, false if it is invalid.

operator=()

Purpose Disposes array's existing data and copies data of the specified array. See comments for CopyArray() for detailed information about how the copy is done.

Access Public

Prototype

```
LArray& operator=(
     const LArray &inArray);
```

Parameters This method takes the following parameter:

`const LArray&`	`inArray`	The reference to the array.

Return A reference to the array.

mComparator

Purpose This data member stores the comparator for the array.

Access Protected

Prototype

```
LComparator *mComparator;
```

mDataAllocated

Purpose Stores the data allocated.

Access Protected

Prototype `Uint32 mDataAllocated;`

mDataStored

Purpose Stores the amount of space to allocate for storage.

Access Protected

Prototype `Uint32 mDataStored;`

mIsSorted

Purpose Indicates whether the array is supposed to be sorted.

Access Protected

Prototype `Boolean mIsSorted;`

mItemCount

Purpose The number of items in the array.

Access Protected

Prototype `Uint32 mItemCount;`

mItemsH

Purpose The handle to the array items.

Access Protected

Prototype `Handle mItemsH;`

mItemSize

Purpose The size of the items.

Access Protected

Prototype
```
Uint32 mItemSize;
```

mIteratorListHead

Purpose The head for the iterator for the array.

Access Protected

Prototype
```
mutable LArrayIterator *mIteratorListHead;
```

mKeepSorted

Purpose Indicates whether the array is to be kept sorted.

Access Protected

Prototype
```
Boolean mKeepSorted;
```

mLockCount

Purpose A reference counter to count the number of times the array is locked (decremented by Unlock()).

Access Protected

Prototype
```
mutable Uint32 mLockCount;
```

mOwnsComparator

Purpose This indicates whether the array owns the comparator.

Access Protected

Prototype

```
Boolean mOwnsComparator;
```

LArrayIterator

Description

LArrayIterator is a PowerPlant class that is useful for iterating through arrays of items.

Methods

The methods in this class are:

LArrayIterator()	~LArrayIterator()
ArrayDied()	CalcNextIndex()
CalcPreviousIndex()	Current()
GetCurrentIndex()	GetNextIterator()
ItemsInserted()	ItemsRemoved()
Next()	Previous()
PtrToCurrent()	PtrToNext()
PtrToPrevious()	ResetTo()
SetNextIterator()	

Data Members

The data members in this class are:

mNextIterator	mArray
mCurrIndex	mNextIndex

Operation

LArrayIterator provides the functionality needed to walk through an array from an arbitrary starting point, going forward or backward through the array elements. Each LArrayIterator object is associated with a single array. An array may have an arbitrary number of iterators, but each iterator operates on only one array.

To learn more about using iterators, refer to *The PowerPlant Book* with the rest of the CodeWarrior documentation.

Index values are signed, 32-bit integers.

Source files

(Array Classes)

`LArrayIterator.h`

`LArrayIterator.cp`

See also LArray

LArrayIterator()

Purpose The constructor creates an object with the passed-in parameters. It constructs an iterator for an array starting at a particular position.

Access Public

Prototype

```
LArrayIterator( const LArray &inArray,
     ArrayIndexT inPosition = from_Start );
```

Parameters The parameters for this constructor are:

`const LArray&`	`inArray`	The reference to the array to iterate on.
`ArrayIndexT`	`inPosition`	The position in the array to set the iterator to. The default is `from_Start`.

~LArrayIterator()

Purpose The destructor destroys the object.

Access Virtual, Public

Prototype

```
~LArrayIterator();
```

ArrayDied()

Purpose This method sets the values of the mCurrIndex and mNextIndex data members to the constant value `index_ArrayDied`, indicating that the array was deleted.

Access Protected

Prototype `void ArrayDied();`

Parameters None

Return None

CalcNextIndex()

Purpose This method calculates the mNextIndex value based on the value of mCurrIndex. If the index is at the end, the value `index_AfterEnd` is assigned.

Access Protected

Prototype `void CalcNextIndex();`

Parameters None

Return None

CalcPreviousIndex()

Purpose This method adjusts the internal indexes to access the previous item in an array.

Access Protected

Prototype `void CalcPreviousIndex();`

Parameters None

Return None

Current()

Purpose This method retrieves the current item in the array and (optionally) the item's size.

Access Public

Prototype

```
Boolean Current( void *outItem,
     Uint32 &ioItemSize );
Boolean Current( void *outItem );
```

Parameters The parameters for these methods are:

`void*`	`outItem`	The pointer to the array item that is passed out upon exiting this method.
`Uint32&`	`ioItemSize`	A reference to the size of the item retrieved from the array.

Return Returns true if the current item exists.

Returns false if the current item does not exist, which happens when:

- Current item was deleted
- Current item is past end of Array
- Current item is before beginning of Array
- Array was deleted

Remarks You should use the `ioItemSize` version of this method with arrays having variable-sized elements.

GetCurrentIndex()

Purpose This method returns the value of the mCurrIndex data member.

Access Inline, Public

Prototype

```
ArrayIndexT GetCurrentIndex() const;
```

Parameters None

Return The value of mCurrIndex.

GetNextIterator()

Purpose This method returns the value of the mNextIterator data member.

Access Inline, Public

Prototype

```
LArrayIterator* GetNextIterator();
```

Parameters None

Return The value of mNextIterator.

ItemsInserted()

Purpose This method keeps track of items that have been inserted into the array at the specified index.

Access Protected

Prototype

```
void ItemsInserted( UInt32 inCount,
     ArrayIndexT inAtIndex );
```

Parameters The parameters for this method are:

`UInt32`	`inCount`	The number of items inserted.
`ArrayIndexT`	`inAtIndex`	The index that is currently pointed to.

Return None

ItemsRemoved()

Purpose This method keeps track of items starting at the specified index that have been removed from the array.

Access Protected

Prototype

```
void ItemsRemoved( UInt32 inCount,
     ArrayIndexT inAtIndex );
```

Parameters The parameters for this method are:

`UInt32`	`inCount`	The number of items removed.
`ArrayIndexT`	`inAtIndex`	The index that is currently pointed to.

Return None

Next()

Purpose This method moves to the next item in the array and passes back a copy of that item and (optionally) the item's size.

Access Public

Prototype

```
Boolean Next( void *outItem, Uint32 &ioItemSize );
Boolean Next( void *outItem );
```

Parameters The parameters for these methods are:

`void*`	`outItem`	The pointer to the array item that is passed out upon exiting this method.
`Uint32&`	`ioItemSize`	A reference to the size of the item retrieved from the array.

Return Returns true if the next item exists

Returns false if next item does not exist, which happens when:

- Current item is at or past end of the Array
- Array was deleted

Previous()

Purpose This method moves to the previous item in the array and passes back a copy of that item.

Access Public

Prototype

```
Boolean Previous( void *outItem, Uint32
     &ioItemSize);
Boolean Previous( void *outItem );
```

Parameters The parameters for these methods are:

`void*`	`outItem`	The pointer to the array item that is passed out upon exiting this method.
`Uint32&`	`ioItemSize`	A reference to the size of the item retrieved from the array.

Return Returns true if the previous item exists.

Returns false if the previos item does not exist, which happens when:

- Current item is at or before the start of the Array
- Array was deleted

PtrToCurrent()

Purpose Return a pointer to the current item in the array and (optionally) pass back the item's size.

Access Public

Prototype

```
void* PtrToCurrent( Uint32 &outItemSize );
void* PtrToCurrent();
```

Parameters The parameter for these methods is:

`Uint32&`	`outItemSize`	A reference to the size of the item retrieved from the array.

Return Returns a pointer to the current item. Returns nil and item size of zero if there is no current item.

PtrToNext()

Purpose Return a pointer to the next item in the array and (optionally) pass back the item's size.

Access Public

Prototype

```
void* PtrToNext( Uint32 &outItemSize );
void* PtrToNext();
```

Parameters The parameter for these methods is:

`Uint32&`	`outItemSize`	A reference to the size of the item retrieved from the array.

Return Returns a pointer to the next item. Returns nil and item size of zero if there is no next item.

PtrToPrevious()

Purpose Return a pointer to the previous item in the array and (optionally) pass back the item's size.

Access Public

Prototype

```
void* PtrToPrevious( Uint32 &outItemSize );
void* PtrToPrevious();
```

Parameters The parameter for these methods is:

`Uint32&`	`outItemSize`	A reference to the size of the item retrieved from the array.

Return Returns a pointer to the previous item. Returns nil and item size of zero if there is no previous item.

ResetTo()

Purpose Reset the current item to the specified index value. The standard enumeration constants `from_Start` and `from_End` are recognized.

Access Public

Prototype

```
void ResetTo( ArrayIndexT inPosition );
```

Parameters The parameter for this method is:

`ArrayIndexT`	`inPosition`	The position to set to, using the enumerations of `from_End`, `index_AfterEnd`, etc.

Return None

SetNextIterator()

Purpose This method set the value of the mNextIterator data member to the value passed-in.

Access Protected

Prototype

```
void SetNextIterator( LArrayIterator *inIterator );
```

Parameters The parameter for this method is:

`LArrayIterator*`	`inIterator`	The value to assign to mNextIterator.

Return None

mNextIterator

Purpose This data member is used to maintain a singly-linked list of Iterators for a particular Array. LArray traverses this linked list of Iterators at various times.

An Array needs a list of all its current Iterators in order to update the Iterators when the Array changes (adding/removing items or even deleting the Array). This keeps the Iterators in synch with the Array.

Access Protected

Prototype

```
LArrayIterator *mNextIterator;
```

mArray

Purpose This data member stores a reference to the array of elements that are iterated over.

Access Protected

Prototype

```
const LArray &mArray;
```

mCurrIndex

Purpose This data member stores the index of the array element currently pointed to.

Access Protected

Prototype

```
ArrayIndexT mCurrIndex;
```

mNextIndex

Purpose This data member stores the index of the array element next in the array.

Access Protected

Prototype

```
ArrayIndexT mNextIndex;
```

LAttachable

Overviedw LAttachable is a PowerPlant class that is used for storing Attachment information in lists.

Methods The methods in this class are:

LAttachable()	~LAttachable()
AddAttachment()	ExecuteAttachments()
GetDefaultAttachable()	RemoveAllAttachments()
RemoveAttachment()	SetDefaultAttachable()

Data Members The data members in this class are:

sDefaultAttachable	mAttachments

Operation Typically, you don't need to override any methods in this class. As it exists, the class provides all the implementation that you are likely to need.

Source files (Feature Classes)

`LAttachable.h`

`LAttachable.cp`

See Also LAttachment

LAttachable()

Purpose The copy constructor initializes the mAttachments data member, but does not make a copy of any other member data. The default constructor initializes the mAttachments data member, and also calls SetDefaultAttachable() with a pointer to this object.

Access Public

Prototypes

```
LAttachable();
```

```
LAttachable::LAttachable(
     const LAttachable& inOriginal );
```

Parameters

`const LAttachable&`	`inOriginal`	This is a reference to the LAttachable object you wish to copy

~LAttachable()

Purpose The destructor destroys the LAttachable object, using a call to RemoveAllAttachments().

Access Public, Virtual

Prototype

```
virtual ~LAttachable();
```

AddAttachment()

Purpose This method adds an Attachment to the list.

Access Public, Virtual

Prototype

```
virtual void AddAttachment(
     LAttachment *inAttachment,
     LAttachment *inBefore,
     Boolean inOwnsAttachment)
```

Parameters This method has the following parameters:

`LAttachment*`	inAttachment	A pointer to the Attachment to add to the list.
`LAttachment*`	inBefore	A pointer to the attachment to precede in the list. Set this to `nil` to add to the end of the list.
Boolean	inOwnsAttachment	If true, the Attachable assumes ownership of the Attachment and will delete the Attachment when the Attachable itself is deleted.

Return None

ExecuteAttachments()

Purpose This method tells all associated Attachments to execute themselves for the specified message.

Access Virtual, Public

Prototype

```
Boolean ExecuteAttachments( MessageT inMessage,
     void *ioParam );
```

Parameters This method has the following parameters:

`MessageT`	`inMessage`	The message that is passed from the framework.
`void*`	`ioParam`	A pointer to the data block that accompanies the message.

Return The `Boolean` return value specifies whether the default Host action should be executed. The value is `false` if any Attachment's Execute() function returns `false`, otherwise it's `true`.

GetDefaultAttachable()

Purpose This method returns a pointer to the sDefaultAttachable data member.

Access Static, Public, Inline

Prototype

```
static LAttachable* GetDefaultAttachable()
```

Parameters None

Return A pointer to sDefaultAttachable.

RemoveAllAttachments()

Purpose This method removes all Attachments from an Attachable. All Attachments owned by this Attachable are deleted.

Access Virtual, Public

Prototype
```
void RemoveAllAttachments();
```

Parameters None

Return None

RemoveAttachment()

Purpose This method removes an Attachment from an Attachable. If this Attachable is the owner of the Attachment, the Attachment's owner is set to `nil`, and the caller should assume control of the Attachment.

Access Virtual, Public

Prototype
```
void RemoveAttachment( LAttachment *inAttachment );
```

Parameters This method has the following parameter:

`LAttachment* inAttachment`	The pointer to the Attachment to remove.

Return None

SetDefaultAttachable()

Purpose This method sets the pointer for the sDefaultAttachable data member.

Access Static, Public, Inline

Prototype
```
static void SetDefaultAttachable(
    LAttachment* inAttachment );
```

Parameters This method has the following parameter:

`LAttachment*`	`inAttachment`	The pointer to the Attachment.

Return None

sDefaultAttachable

Purpose This data member contains the pointer to the default Attachable object.

Access Protected

Prototype

```
static LAttachable* sDefaultAttachable;
```

mAttachments

Purpose This data member is a list of all the Attachments.

Access Protected

Prototype

```
TArray<LAttachment*> *mAttachments;
```

LAttachment

Description

LAttachment is a PowerPlant class that is used for forming re-lataionships between objects. An Attachment is usually an object that modifies the runtime behavior of another object.

Methods

The methods in this class are:

LAttachment()	~LAttachment()
Execute()	ExecuteSelf()
GetExecuteHost()	GetMessage()
GetOwnerHost()	SetExecuteHost()
SetMessage()	SetOwnerHost()

Data Members

The data members in this class are:

mOwnerHost	mMessage
mExecuteHost	

Operation

The concept of an Attachment is a powerful underlying concept of PowerPlant's inner workings. For a detailed discussion on this topic, refer to *The PowerPlant Book*.

Source files

(Feature Classes)

`LAttachment.h`

`LAttachment.cp`

See also

LAttachable

LPaintAttachment

LYieldAttachment

LAttachment()

Purpose The constructors create the object and initialize the data members to the passed-in values.

Access Public

Prototype

```
LAttachment( MessageT inMessage,
     Boolean inExecuteHost);
LAttachment( LStream *inStream );
```

Parameters The parameters for the constructors are:

`MessageT`	`inMessage`	The value to set mMessage to.
`Boolean`	`inExecuteHost`	The value to set mExecuteHost to.
`LStream*`	`inStream`	The data in Stream must be: • MessageT—Message that Attachment responds to. • Boolean—Should we execute the Host? (mExecuteHost) • Boolean—Does the Host own this object? (mOwnerHost)

~LAttachment()

Purpose The destructor destroys the object, removing itself from any Attachments.

Access Virtual, Public

Prototype

```
virtual ~LAttachment();
```

Parameters None

Execute()

Purpose If the message passed to this method is the message for which the Attachment is designed, this method calls ExecuteSelf().

Access Virtual, Public

Prototype

```
virtual Boolean Execute( MessageT inMessage,
     void *ioParam);
```

Parameters This method has the following parameters:

`MessageT`	`inMessage`	The message to which the attachment responds.
`void*`	`ioParam`	A pointer to the host attachable object.

Return A `Boolean` with the value of mExecuteHost. The PowerPlant framework uses this value to decide whether the host object should also perform the task in question.

Remarks You do not typically need to override this function.

ExecuteSelf()

Purpose This method performs the attachment task. You typically want to override this method in your class that inherits from LAttachment.

Access Virtual, Public

Prototype

```
void ExecuteSelf(MessageT inMessage,
     void* ioParam );
```

Parameters This method has the following parameters:

`MessageT`	`inMessage`	The message to which the attachment responds.
`void*`	`ioParam`	A pointer to the host attachable object.

Return None

Remarks To get an idea of how to implement this method in your class override, look at classes like LPaintAttachment to see how it is implemented.

GetExecuteHost()

Purpose This method is an accessor for the value of mExecuteHost.

Access Inline, Public

Prototype

```
Boolean GetExecuteHost();
```

Parameters None

Return A `Boolean` indicating the value of mExecuteHost.

GetMessage()

Purpose This method is an accessor for the value of mMessage.

Access Public, Inline

Prototype

```
MessageT GetMessage();
```

Parameters None

Return A `MessageT` indicating the value of mMessage.

GetOwnerHost()

Purpose This method is an accessor for the value of mOwnerHost.

Access Public, Inline

Prototype
```
LAttachable* GetOwnerHost();
```

Parameters None

Return An LAttachable pointer indicating the value of mOwnerHost.

SetExecuteHost()

Purpose Sets the value of mExecuteHost, that specifies whether the host action should execute after executing all Attachments.

Access Virtual, Public

Prototype
```
void SetExecuteHost( Boolean inExecuteHost );
```

Parameters This method takes a Boolean to assign to mExecuteHost.

Return None

SetMessage()

Purpose Sets the value of mMessage.

Access Virtual, Public

Prototype
```
void SetMessage( MessageT inMessage );
```

Parameters A `MessageT` to assign to mMessage.

Return None

SetOwnerHost()

Purpose Sets the value of mOwnerHost.

Access Virtual, Public

Prototype
```
void SetOwnerHost( LAttachable *inHost );
```

Parameters A pointer to an LAttachable to assign to mOwnerHost.

Return None

mOwnerHost

Purpose This data member points to the host attachable object.

Access Protected

Prototype
```
LAttachable *mOwnerHost;
```

mMessage

Purpose This data member contains the message value to which the attachable object responds.

Access Protected

Prototype
```
MessageT mMessage;
```

mExecuteHost

Purpose This data member indicates whether or not to execute the host object action.

Access Protected

Prototype
```
Boolean mExecuteHost;
```

LBeepAttachment

Overview LBeepAttachment is a PowerPlant class that is used for beeping when a message is received.

Methods The methods in this class are:

LBeepAttachment() ExecuteSelf()

Data Members There are no data members in this class.

Operation This object is a simple Attachment designed to respond to any message you indicate, typically a click message. When you create the object with LBeepAttachment(), you specify the message to which you want the Attachment to respond. When attached to any host, this Attachment beeps when the appropriate message is received.

Source files (Utility Classes)

`UAttachments.h`

`UAttachments.cp`

See also LAttachment

LPane

LBeepAttachment()

Purpose The constructor for this object is a placeholder.

Access Public

Prototype

```
LBeepAttachment( MessageTinMessage,
     Boolean inExecuteHost ) ;
LBeepAttachment( LStream *inStream);
```

Parameters The parameters for these constructors are:

`MessageT`	`inMessage`	The message that we respond to.
`LStream*`	`inStream`	A pointer to a stream object that contains the information to create the LBeepAttachment object.

ExecuteSelf()

Purpose This method beeps using `SysBeep`(). This is an override of Execute-Self() in LAttachment.

LBorderAttachment

Overview LBorderAttachment is a PowerPlant class that is used for drawing a border within the Frame of a Pane. It is often used in conjunction with LPaintAttachment.

Methods The methods in this class are:

LBorderAttachment() ExecuteSelf()

Data Members The data members in this class are:

mPenState mForeColor

mBackColor

Operation This Attachment is designed to respond to the `msg_DraworPaint` message. When you create this Attachment, you specify a `PenState`, foreground and background colors, and whether the host should also draw.

Source files (Utility Classes)

`UAttachments.h`

`UAttachments.cp`

See also LAttachment

LPaintAttachment

LPane

LBorderAttachment()

Purpose The constructor creates the object containing the passed-in parameters.

Access Public

Prototype

```
LBorderAttachment( PenState *inPenState,
     RGBColor *inForeColor,
     RGBColor *inBackColor,
     Boolean inExecuteHost );
LBorderAttachment( LStream *inStream );
```

Parameters The parameters for these constructors are:

`PenState*`	`inPenState`	A pointer to a PenState.
`RGBColor*`	`inForeColor`	A pointer to an RGBColor.
`RGBColor*`	`inBackColor`	A pointer to an RGBColor.
Boolean	`inExecuteHost`	A Boolean indicating the value to be set for mExecuteHost.
`LStream*`	`inStream`	A pointer to a stream object that contains the information to create the LBorderAttachment object.

ExecuteSelf()

Purpose This method draws a rectangular border. This is an override of ExecuteSelf() in LAttachment.

mPenState

Purpose This member holds the pen information.

Access Protected

Prototype

```
PenState mPenState;
```

mForeColor

Purpose This member holds the foreground color information.

Access Protected

Prototype `RGBColor mForeColor;`

mBackColor

Purpose This member holds the background color information.

Access Protected

Prototype `RGBColor mBackColor;`

LBroadcaster

Overview

LBroadcaster is a PowerPlant class that is used for creating Broadcast and Listen relationships between objects. A broadcaster object communicates to a Listener object by sending messages. This is udeful if you want to have an object that can send messages to many other objects.

Methods

The methods in this class are:

LBroadcaster()	~LBroadcaster()
AddListener()	BroadcastMessage()
IsBroadcasting()	RemoveListener()
StartBroadcasting()	StopBroadcasting()

Data Members

The data members in this class are:

mListeners	mIsBroadcasting

Operation

A Broadcaster sends messages to its Listeners. LBroadcaster is an abstract, mix-in class. Broadcasters have a list of Listeners to which they send messages via the BroadcastMessage() method. You attach a Listener to a Broadcaster using the AddListener() function.

Classes derived from LBroadcaster should call the BroadcastMessage() method whenever they want to announce some happening in the code, such as a change in state. For example, the destructor for LBroadcaster sends a `msg_BroadcasterDied` message to its Listeners.

Source files

(Feature Classes)

`LBroadcaster.h`

`LBroadcaster.cp`

See also

LListener

LBroadcaster()

Purpose The copy constructor for LBroadcaster makes a copy of the state of mIsBroadcasting data member. Listener links are not copied. The default constructor has no Listeners initially.

Access Public

Prototype
```
LBroadcaster();
LBroadcaster(const LBroadcaster& inOriginal);
```

Parameters None

Return No return value for a constructor

~LBroadcaster()

Purpose The destructor destroys the LBroadcaster object, and notifies all Listeners that this Broadcaster object is going away.

Access Public, Virtual

Prototype
```
virtual ~LBroadcaster();
```

Parameters None

Return None

AddListener()

Purpose This method adds a Listener to a Broadcaster object. You must associate Broadcasters with Listeners, not the converse:

```
theBroadcaster->AddListener(theListener); // correct
theListener->AddBroadcaster(theBroadcaster); // incorrect!
```

This method takes care of notifying the Listener to update its list of Broadcasters.

Access Public

Prototype

```
void AddListener( LListener *inListener );
```

Parameters This method takes the following parameter:

`LListener*`	inListener	a pointer to the LListener object that will be associated with the LBroadcaster

Return None

BroadcastMessage()

Purpose This method broadcasts a message to all associated Listeners. Listeners are associated with a Broadcaster using AddListener(). This method does not send a message to its Listeners if broadcasting is turned off. Broadcasting is turned on using StartBroadcasting(), and can be turned off using StopBroadcasting().

Access Public

Prototype

```
void BroadcastMessage(MessageT inMessage,
      void *ioParam = nil);
```

Parameters This method takes the following parameters:

MessageT	inMessage	This is the message you want to send to all Listeners.
void	*ioParam	This is a pointer to a parameter structure that the Listeners may use. The default value is nil, meaning no parameter.

Return None

Remarks

- Does not send message if broadcasting is turned off via StopBroadcasting().

- The meaning of the `ioParam` parameter depends on the message.

IsBroadcasting()

Purpose This method tells you whether broadcasting is turned on or not.

Access Public, Inline

Prototype
```
Boolean IsBroadcasting();
```

Parameters None

Return A Boolean indicating "true" if broadcasting is turned on, or "false" otherwise.

RemoveListener()

Purpose This method tells a Listener to remove itself from the broadcast list. In response to this, a Listener should update its list of Broadcasters.

Access Public

Prototype
```
void RemoveListener( LListener *inListener );
```

Parameters This method takes the following parameter:

LListener*	inListener	A pointer to the Listener object that we want to remove from the broadcast list.

Return None

StartBroadcasting()

Purpose This method enables broadcasting, by setting the mIsBroadcasting data member to true.

Access Public, Inline

Prototype `void StartBroadcasting();`

Parameters None

Return None

StopBroadcasting()

Purpose This method disables broadcasting, by setting the mIsBroadcasting data member to false. A Broadcaster that has broadcast turned off does not broadcast messages.

Access Public, Inline

Prototype `void StopBroadcasting();`

Parameters None

Return None

mListeners

Purpose This data member is a list of pointers to LListener objects. In general, you should use the AddListener() and RemoveListener() methods to access this list.

Access Protected

Prototype `TArray<LListener*> mListeners;`

mIsBroadcasting

Purpose This data member indicates whether broadcasting is enabled or not. This member should be accessed using the IsBroadcasting() method.

Access Protected

Prototype `Boolean mIsBroadcasting;`

LButton

Overview

LButton is a PowerPlant class that implements a button. The graphic for the button needs to be stored as a resource of one of these types:

- ICN#
- ICON
- PICT

Methods

The methods in this class are:

LButton()	DrawGraphic()
DrawSelf()	HotSpotAction()
HotSpotResult()	PointIsInFrame()
SetGraphics()	SetGraphicsType()

Data Members

The data members in this class are:

mGraphicsType	mNormalID
mPushedID	

Operation

When you create an object of this type, you provide two resource ID numbers. One is for the graphical element in its non-depressed state, and the other is for the depressed (user-pushed) state.

If you use the `ICN#` resource for the button, the Mac OS automatically picks the icon family member that best matches the display settings of the monitor on which it is rendered.

Although buttons are usually small, you can use a large `PICT` as a button. Note that `ICON` and `ICN#` resources describe images that are of Mac OS-standard, specific dimensions.

When the user releases the mouse button with the cursor inside the button perimeter, the button sends a message to its Listeners.

Source files

(Pane Classes)

LButton.h

LButton.cp

Ancestors LBroadcaster

LControl

LPane

LButton()

Purpose The constructors create new objects from the passed-in parameters.

Access Public

Prototype

```
LButton();
LButton( const LButton &inOriginal );
LButton( const SPaneInfo &inPaneInfo,
     MessageT inClickedMessage,
     OSType inGraphicsType,
     ResIDT inNormalID,
     ResIDT inPushedID );
LButton( LStream *inStream );
```

Parameters These constructors have the following parameters:

`const LButton&`	`inOriginal`	A reference to the LButton object you want to copy.
`const SPaneInfo&`	`inPaneInfo`	A reference to the SPaneInfo object that is the super view.
`MessageT`	`inClickedMessage`	The message sent when the button is clicked.
`OSType`	`inGraphicsType`	Graphics Type ('ICN#', 'ICON', or 'PICT')
`ResIDT`	`inNormalID`	Resource ID for normal graphic
`ResIDT`	`inPushedID`	Resource ID for pushed graphic
`LStream*`	`inStream`	A pointer to a stream object that contains the information to create the LButton object.

DrawGraphic()

Purpose Draw the graphic for a button. The pane must already be focused; refer to LPane for more information.

Access Virtual, Protected

Prototype

```
virtual void DrawGraphic( ResIDT inGraphicID );
```

Parameters The parameter for this method is:

`ResIDT`	`inGraphicID`	The resource ID of the graphic to draw in the button.

Return None

DrawSelf()

Purpose This method is an override of the base class method DrawSelf() in LPane. It draws the normal graphic in the button.

HotSpotAction()

Purpose This method is an override of the base class method HotSpotAction() in LControl. It causes the buttons to toggle between two graphics, depending on whether the mouse is inside or outside the button.

HotSpotResult()

Purpose This method is an override of the base class method HotSpotResult() in LControl. It broadcasts a message to Listeners when the button is clicked.

PointIsInFrame()

Purpose This method is an override of the base class method PointIsInFrame() in LPane. It gives you information about whether a point lies within a given frame.

SetGraphics()

Purpose Sets the graphic resources to use for drawing the button.

Access Virtual, Public

Prototype

```
virtual void SetGraphics( ResIDT inNormalID,
     ResIDT inPushedID);
```

Parameters This method has the following parameters:

`ResIDT`	`inNormalID`	Resource ID for normal graphic
`ResIDT`	`inPushedID`	Resource ID for pushed graphic

Return None

SetGraphicsType()

Purpose This method sets the value of the mGraphicsType data member.

Access Virtual, Public

Prototype

```
virtual void SetGraphicsType(
     OSType inGraphicsType );
```

Parameters This method has the following parameters:

`OSType`	`inGraphicsType`	The value to set for mGraphicsType.

Return None

mGraphicsType

Purpose This method holds the resource type descriptor for the graphic elements of the button. It should be one of the following:

- ICN#
- ICON'
- 'PICT'

Access Protected

Prototype
```
OSType mGraphicsType;
```

mNormalID

Purpose This data member holds the resource ID for the normal button graphic.

Access Protected

Prototype
```
ResIDT mNormalID;
```

mPushedID

Purpose This data member holds the resource ID for the pushed button graphic.

Access Protected

Prototype
```
ResIDT mPushedID;
```

LCaption

Overview

LCaption is a PowerPlant class that is used for displaying static text.

Methods

The methods in this class are:

LCaption()	~LCaption()
DrawSelf()	GetDescriptor()
GetTextTraitsID()	GetValue()
SetDescriptor()	SetTextTraitsID()
SetValue()	

Data Members

The data members in this class are:

mText	mTxtrID

Operation

This class uses a text traits resource to specify characteristics such as font, size, style, color, and justification. LCaption uses the UTextDrawing class to draw text.

You can configure the text and the text traits resource in the Constructor resource editor, or you can may these characteristics at runtime.

It is possible that you will encounter a drawing problem if you modify the contents of the caption at runtime. DrawSelf() uses the DrawWithJustification() method of UTextDrawing, that does not erase the previous contents of the caption. The best way to erase the contents is to attach an LEraseAttachment object to the caption.

Source files

(Pane Classes)

`LCaption.h`

`LCaption.cp`

See also

LAttachment

LEraseAttachment

UTextDrawing

LCaption()

Purpose The constructors create the object from the passed-in parameters.

Access Public

Prototype

```
LCaption();
LCaption( const LCaption &inOriginal );
LCaption( const SPaneInfo &inPaneInfo,
     ConstStringPtr inString,
     ResIDT inTextTraitsID );
LCaption( LStream *inStream );
```

Parameters The parameters for these constructors are:

`SPaneInfo&`	`inPaneInfo`	A reference to the Pane that hosts this caption.
`LCaption&`	`inOriginal`	A reference to a caption to be copied.
`ConstStringPtr`	`inString`	A pointer to the caption text.
`ResIDT`	`inTextTraitsID`	The text traits for the caption.
`LStream*`	`inStream`	A pointer to a stream object that contains the information to create the LCaption object.

~LCaption()

Purpose The destructor destroys the object.

Access Virtual, Public

Prototype

```
virtual ~LCaption();
```

Parameters None

DrawSelf()

Purpose Draw the caption using the colors and traits specified.

Access Virtual, Protected

Prototype
```
virtual void DrawSelf();
```

Parameters None

Return None

GetDescriptor()

Purpose Return contents of the caption as a string.

Access Virtual, Public

Prototype
```
virtual StringPtr GetDescriptor(
     Str255 outDescriptor) const;
```

Parameters The parameter for this method is:

`Str255`	`outDescriptor`	A pointer to a buffer, to contain the string on exit.

Return A pointer to a string, using the `StringPtr` type.

GetTextTraitsID()

Purpose This method returns the value of the mTxtrID data member.

Access Inline, Public

Prototype
```
ResIDT GetTextTraitsID() const;
```

Parameters None

Return A resource ID, using the `ResIDT` type.

GetValue()

Purpose Return the integer value represented by the text of the caption, held in the mText data member.

Access Virtual, Public

Prototype
```
virtual Int32 GetValue() const;
```

Parameters None

Return `Int32` containing the value of mText.

SetDescriptor()

Purpose Set contents of the caption from a string.

Access Virtual, Public

Prototype
```
virtual void SetDescriptor(
     ConstStringPtr inDescriptor);
```

Parameters The parameter for this method is:

`ConstStringPtr`	`inDescriptor`	A pointer to a buffer, containing the string to set for the caption.

Return None

SetTextTraitsID()

Purpose This member sets the value of the mTxtrID data member to a resource ID for a resource containing the text traits information.

Access Public

Prototype `void SetTextTraitsID( ResIDT inTxtrID );`

Parameters The parameter for this method is:

`ResIDT inTxtrID`	A `ResIDT` containing the resource ID value to put in the mTxtrID data member.

Return None

SetValue()

Purpose Set a caption to the text representation of an integer value.

Access Virtual, Public

Prototype `virtual void SetValue( Int32 inValue );`

Parameters The parameter for this method is:

`Int32 inValue`	The integer value to set the caption to.

Return None

mText

Purpose This data member contains the caption string.

Access Protected

Prototype `LStr255 mText;`

mTxtrID

Purpose This data member contains a resource ID for a resource that contains the text traits information for the caption.

Access Protected

Prototype `ResIDT mTxtrID;`

LCicnButton

Overview

LCicnButton is a PowerPlant class that is very similar to LButton. It is different from LButton in that it uses the Mac OS `'cicn'` color icon resource format as the graphical element in the button.

Methods

The methods in this class are:

LCicnButton()	~LCicnButton()
DrawSelf()	HotSpotAction()
HotSpotResult()	SetCicns()

Data Members

The data members in this class are:

mNormalID	mPushedID
mNormalCicnH	mPushedCicnH

Operation

As with the LButton class, you provide resource ID numbers for the button in both its normal and pushed states.

Source files

(Pane Classes)

`LCicnButton.h`

`LCicnButton.cp`

See also

LButton

LControl

LCicnButton()

Purpose

The constructors create new objects from the passed-in parameters.

Access

Public

Prototype

```
LCicnButton();
LCicnButton( const LCicnButton &inOriginal );
```

```
LCicnButton( const SPaneInfo &inPaneInfo,
     MessageT inClickedMessage,
     ResIDT inNormalID,
     ResIDT inPushedID );
LCicnButton( LStream *inStream );
```

Parameters These constructors have the following parameters:

`const LCicnButton&`	`inOriginal`	A reference to the object you want to copy.
`const SPaneInfo&`	`inPaneInfo`	A reference to the SPaneInfo object that is the
`MessageT`	`inClicked-Message`	The message sent when the button is depressed.
`ResIDT`	`inNormalID`	Resource ID for normal graphic
`ResIDT`	`inPushedID`	Resource ID for pushed graphic
`LStream*`	`inStream`	A pointer to a stream object that contains the information to create the LButton object.

~LCicnButton()

Purpose The destructor destroys the object.

Access Virtual, Public

Prototype

```
virtual ~LCicnButton();
```

Parameters None

DrawSelf()

Purpose This method behaves just like the one in LCicnButton() named DrawSelf(), except it draws a `'cicn'` button instead.

HotSpotAction()

Purpose This method behaves just like the one in LCicnButton() named HotSpotAction(), except it works for a `'cicn'` button instead.

HotSpotResult()

Purpose This method behaves just like the one in LCicnButton() named HotSpotResult(), except it works for a `'cicn'` button instead.

SetCicns()

Purpose Specify new ID's for the normal and pushed 'cicn' resources.

Access Virtual, Public

Prototype

```
virtual void SetCicns( ResIDT inNormalID,
     ResIDT inPushedID);
```

Parameters The parameters for this method are:

`ResIDT`	`inNormalID`	The resource ID for the normal graphic.
`ResIDT`	`inPushedID`	The resource ID for the pushed graphic.

Return None

mNormalID

Purpose This data member is just like the one in LButton named mNormalID, except it is for `'cicn'` buttons.

mPushedID

Purpose This data member is just like the one in LButton named mPushedID, except it is for `'cicn'` buttons.

mNormalCicnH

Purpose This data member holds a handle to the resource for the normal `'cicn'`.

Access Protected

Prototype

```
CIconHandle mNormalCicnH;
```

mPushedCicnH

Purpose This data member holds a handle to the resource for the pushed `'cicn'`.

Access Protected

Prototype

```
CIconHandle mPushedCicnH;
```

LCleanupTask

Overview

LCleanupTask is a PowerPlant class that is an abstract base class for any operation which needs to be performed at application shut-down time. It patches `ExitToShell()` to ensure that tasks are performed even if the user force-quits application.

Methods

The methods in this class are:

LCleanupTask()
~LCleanupTask()
CleanUpAtExit()
DoCleanup()
ETSPatch()

Data Members

The data members in this class are:

sCleanupTaskList
sOldETSRoutine
sNewETSRoutine

Operation

This class may be used as a mix-in to ensure that an object cleans up after itself when the application quits.

Source files

(Networking Classes)

`LCleanupTask.h`

`LCleanupTask.cp`

See also

LInterruptSafeList

LMacTCPTCPEndpoint

LOpenTptUDPEndpoint

LMacTCP_CloseResolver

LOpenTpt_CloseOpenTpt

LCleanupTask()

Purpose The constructor creates the object. Don't use the copy contructor.

Access Protected

Prototype
```
LCleanupTask();
LCleanupTask(LCleanupTask&); /* copy ctor */
```

Parameters None

~LCleanupTask()

Purpose The destructor destroys the LCleanupTask object.

Access Virtual, Protected

Prototype
```
~LCleanupTask();
```

CleanUpAtExit()

Purpose Should be called only once whenever the application quits, either by the normal Quit command exit or by a user abort. This method runs all of the cleanup tasks and removes them from the list.

Access Static, Public

Prototype
```
void CleanUpAtExit();
```

Parameters None

Return None

DoCleanup()

Purpose Override to perform any task that must be performed before the app quits.

Access Virtual, Protected

Prototype `void DoCleanup();`

Parameters None

Return None

ETSPatch()

Purpose Patch `ExitToShell()`.

Access Static, Protected

Prototype `void ETSPatch();`

Parameters None

Return None

sCleanupTaskList

Purpose The task list for cleanup.

Access Static, Protected

Prototype `LInterruptSafeList*sCleanupTaskList;`

sOldETSRoutine

Purpose Temporary storage for the `ExitToShell()` routine.

Access Static, Protected

Prototype `UniversalProcPtrsOldETSRoutine;`

sNewETSRoutine

Purpose The pointer to the new `ExitToShell()` routine.

Access Static, Protected

Prototype

```
UniversalProcPtrsNewETSRoutine;
```

LClipboard

Overview

LClipboard is a PowerPlant class that is used for supporting the global clipboard in your application. It contains all the methods for setting and getting data or arbitrary type and size.

Methods

The methods in this class are:

LClipboard()	~LClipboard()
ExecuteSelf()	ExportSelf()
GetClipboard()	GetData()
GetDataSelf()	ImportSelf()
SetData()	SetDataSelf()

Data Members

The data members in this class are:

sClipboard	mImportPending
mExportPending	

Operation

To use LClipboard, you create a single instance of it in your application, and attach the clipboard object to your application object. As an attachment, LClipboard looks for the msg_Event message. If a suspend or resume event occurs, LClipboard converts the local clipboard to the global scrap, or vice versa.

You always get a pointer to the clipboard with the static method GetClipboard(). To learn more information on using this object, refer to *The PowerPlant Book*.

Source files

(Support Classes)

`LClipboard.h`

`LClipboard.cp`

Ancestors

LAttachment

LClipboard()

Purpose The constructor creates the clipboard for your application.

Access Public

Prototype
```
LClipboard();
```

Parameters None

~LClipboard()

Purpose The destructor destroys the clipboardobject.

Access Virtual, Public

Prototype
```
virtual ~LClipboard();
```

ExecuteSelf()

Purpose Clipboard watches for Suspend and Resume events. The `inMessage` parameter will be `msg_Event`. The `ioParam` will be an `EventRecord*`. This method is an override of ExecuteSelf() in LAttachment.

ExportSelf()

Purpose Export the data in a local scrap to the global scrap.

Access Protected

Prototype
```
virtual void ExportSelf();
```

Parameters None

Return None

Remarks This implementation does nothing since this class uses the global scrap when setting and getting clipboard data. Subclasses should override this method if they maintain a local scrap.

GetClipboard()

Purpose This method returns the value of sClipboard, a pointer to the clipboard.

Access Inline, Static, Public

Prototype
```
static LClipboard* GetClipboard();
```

Parameters None

Return A pointer to an `LClipboard` object.

Remarks There is only one `LClipboard` object in an application, since this is a static data member.

GetData()

Purpose This method passes back the data in the Clipboard of the specified type in a Handle, and returns the size of the data.

If `ioDataH` is `nil`, the data is not passed, but its size is returned. Otherwise, `ioDataH` is resized if necessary to hold all the data.

This is a wrapper method which imports the global scrap if necessary, then calls a lower level function to actually retrieve the data.

Access Virtual, Public

Prototype
```
virtual Int32 GetData( ResType inDataType,
     Handle ioDataH);
```

Parameters This method has the following parameters:

`ResType`	`inDataType`	This is the resource ID for the type of data.
`Handle`	`ioDataH`	This is a handle to the data that is resized to accommodate the data.

Return `Int32` indicating the size of the data, in bytes.

GetDataSelf()

Purpose Pass back the data in the Clipboard of the specified type in a Handle and return the size of the data

If ioDataH is nil, the data is not passed data, but its size is returned Otherwise, ioDataH is resized if necessary to hold all the data

This implementation gets the data from the global scrap. Subclasses should override to maintain a local scrap.

Access Virtual, Protected

Prototype

```
virtual Int32 GetDataSelf( ResType inDataType,
    Handle ioDataH);
```

Parameters This method has the following parameters:

`ResType`	`inDataType`	This is the resource ID for the type of data.
`Handle`	`ioDataH`	This is a handle to the data that is resized to accommodate the data.

Return `Int32` indicating the size of the data, in bytes.

ImportSelf()

Purpose This implementation does nothing since this class uses the global scrap when setting and getting clipboard data. Subclasses should override this method if they maintain a local scrap.

Access Virtual, Protected

Prototype
```
virtual void ImportSelf();
```

Parameters None

Return None

SetData()

Purpose This method sets the Clipboard contents to the data specified by a pointer and length, or it sets the Clipboard contents to the data in a Handle.

Access Virtual, Public

Prototype
```
virtual void SetData( ResType inDataType,
     Ptr inDataPtr,
     Int32 inDataLength,
     Boolean inReset );
virtual void SetData( ResType inDataType,
     Handle inDataH,
     Boolean inReset );
```

Parameters This method has the following parameters:

`ResType`	`inDataType`	This is the resource ID for the type of data.
`Ptr`	`inDataPtr`	This is a pointer to some data to put on the Clipboard.
`Handle`	`inDataH`	This is a handle to the data that is resized to accommodate the data.

`Int32`	`inData-Length`	This is the size of the data pointed to by `inDataPtr`.
`Boolean`	`inReset`	This specifies whether to clear the existing contents of the Clipboard before storing the new data.

Return None

SetDataSelf()

Purpose This method sets the Clipboard contents to the data specified by a pointer and length. This implementation sets the data in the global scrap. Subclasses should override this method to maintain a local scrap.

Access Virtual, Protected

Prototype

```
virtual void SetDataSelf( ResType inDataType,
    Ptr inDataPtr,
    Int32 inDataLength,
    Boolean inReset );
```

Parameters This method has the following parameters:

`ResType`	`inDataType`	This is the resource ID for the type of data.
`Ptr`	`inDataPtr`	This is a pointer to some data to put on the Clipboard.
`Int32`	`inData-Length`	This is the size of the data pointed to by `inDataPtr`.
`Boolean`	`inReset`	This specifies whether to clear the existing contents of the Clipboard before storing the new data.

Return None

sClipboard

Purpose This data member is a static pointer to the Clipboard. There is only one Clipboard, hence the reason for the static pointer.

Access Protected

Prototype

```
static LClipboard *sClipboard;
```

mImportPending

Purpose This data member indicates whether an import is pending. If the scrap changed while we were suspended, then we set the import pending flag to true. We don't need to import the data now, since the program may never request the data. This member would be set on receiving a Resume event.

Access Protected

Prototype

```
Boolean mImportPending;
```

mExportPending

Purpose This data member indicates whether an export is pending. This member would be set on receiving a Suspend event.

Access Protected

Prototype

```
Boolean mExportPending;
```

LCmdEnablerAttachment

Overview

LCmdEnablerAttachment is a PowerPlant class that is used for coupling commands and updating chores in PowerPlant object relationships.

Methods

The methods in this class are:

LCmdEnablerAttachment() ExecuteSelf()

Data Members

The data members in this class are:

mCmdToEnable

Operation

This Attachment responds to the `msg_CommandStatus` message. When you create the Attachment, you also specify the command that should be enabled.. When it executes, this Attachment enables the menu item associated with the command. It also prevents the host from executing, because it has already enabled the related command. It does not do any item manipulation such as setting a mark in the menu.

You can uset shi Attachment as a model for handling other updating tasks. For example, you might create a check mark Attachment that puts a check mark in front of a menu item.

Source files

(Utility Classes)

`UAttachments.h`

`UAttachments.cp`

See also

LAttachment

LPane

LCmdEnablerAttachment()

Purpose The constructor creates the object containing the passed-in parameters.

Access Public

Prototype

```
LCmdEnablerAttachment( CommandT inCmdToEnable );
LCmdEnablerAttachment( LStream* inStream );
```

Parameters The parameters for these constructors are:

`CommandT`	`inCmdToEnable`	A command that we want to enable.
`LStream*`	`inStream`	A pointer to a stream object that contains the information to create the LCmdEnablerAttachment object.

ExecuteSelf()

Purpose This method decides whether to enable the command, then sets mExecuteHost appropriately. This is an override of ExecuteSelf() in LAttachment.

mCmdToEnable

Purpose This member holds the value of the command for which we want to control enabling behavior.

Access Protected

Prototype

```
CommandT mCmdToEnable;
```

LColorEraseAttachment

Overview LColorEraseAttachment is a PowerPlant class that is used for erasing the Frame of a Pane.

Methods The methods in this class are:

LColorEraseAttachment() ExecuteSelf()

Data Members The data members in this class are:

mForeColor mBackColor

Operation This Attachment is used with the `msg_DrawOrPrint` message.

Source files (Utility Classes)

`UAttachments.h`

`UAttachments.cp`

See also LAttachment

LPane

LColorEraseAttachment()

Purpose The constructor creates the object containing the passed-in parameters.

Access Public

Prototype

```
LColorEraseAttachment( RGBColor *inBackColor,
     Boolean inExecuteHost);
LColorEraseAttachment( PenState* inPenState,
     RGBColor *inForeColor,
```

```
    RGBColor *inBackColor,
    Boolean inExecuteHost);
LColorEraseAttachment( LStream *inStream );
```

Parameters The parameters for these constructors are:

`PenState*`	`inPenState`	A pointer to a PenState.
`RGBColor*`	`inForeColor`	A pointer to an RGBColor.
`RGBColor*`	`inBackColor`	A pointer to an RGBColor.
Boolean	`inExecuteHost`	A Boolean indicating the value to be set for mExecuteHost.
`LStream*`	`inStream`	A pointer to a stream object that contains the information to create the LColorEraseAttachment object.

ExecuteSelf()

Purpose This method erases a rectangular area. This is an override of ExecuteSelf() in LAttachment.

mForeColor

Purpose This member holds the foreground color information.

Access Protected

Prototype

```
RGBColor mForeColor;
```

mBackColor

Purpose This member holds the background color information.

Access Protected

Prototype

```
RGBColor mBackColor;
```

LCommander

Overview

LCommander is a PowerPlant class that is used for managing the state of menu items while they are the active target or in the active chain of command.

Methods

The methods in this class are:

LCommander()
~LCommander()
AddSubCommander()
AllowSubRemoval()
AllowTargetSwitch()
AttemptQuit()
AttemptQuitSelf()
BeTarget()
DontBeTarget()
FindCommandStatus()
GetDefaultCommander()
GetLatentSub()
GetSuperCommander()
GetTarget()
GetTopCommander()
GetUpdateCommandStatus()
HandleKeyPress()
InitCommander()
IsOnDuty()
IsSyntheticCommand()
IsTarget()
ObeyCommand()
PostAction()
PostAnAction()
ProcessCommand()
ProcessCommandStatus()
ProcessKeyPress()
PutChainOnDuty()
PutOnDuty()
RemoveSubCommander()
RestoreTarget()
SetDefaultCommander()
SetLatentSub()
SetSuperCommander()
SetTarget()
SetUpdateCommandStatus()
SwitchTarget()
TakeChainOffDuty()
TakeOffDuty()

Data Members

The data members in this class are:

sTopCommander	sTarget
sDefaultCommander	sUpdateCommandStatus
mSuperCommander	mSubCommanders
mOnDuty	

Operation

A commander may be on or off duty, and has functions to manage the duty state. This is an important concept, because an off-duty commander will not receive or respond to events.

To learn more about Commanders and how to use them with PowerPlant, refer to *The PowerPlant Book*.

Source files

(Commander Classes)

`LCommander.h`

`LCommander.cp`

Ancestors

LAttachable

See Also

LMenu

LMenuBar

LCommander()

Purpose

The constructor creates objects from the passed-in parameters.

Access

Public

Prototype

```
LCommander();
LCommander( const LCommander &inOriginal );
LCommander( LCommander *inSuper );
```

Parameters The parameters for these constructors are:

`const LCommander&`	`inOriginal`	The reference to the object to copy.
`LCommander*`	`inSuper`	The pointer to the super commander.

~LCommander()

Purpose The destructor destroys the object.

Access Virtual, Public

Prototype
```
virtual ~LCommander();
```

AddSubCommander()

Purpose Adds a subcommander to the commander object.

Access Virtual, Protected

Prototype
```
virtual void AddSubCommander( LCommander *inSub );
```

Parameters The parameter for this method is:

`LCommander*`	`inSub`	The pointer to the subcommander.

Return None

AllowSubRemoval()

Purpose This method indicates whether removal of subcommanders is allowed.

Access Virtual, Public

Prototype
```
virtual Boolean AllowSubRemoval(LCommander* inSub);
```

Parameters The parameter for this method is:

`LCommander*`	`inSub`	Unused.

Return Boolean indicating whether the removal is allowed.

AllowTargetSwitch()

Purpose Grant permission to switch the target to the specified Commander.

Both the current target and `inNewTarget` must be Subordinates of this Commander.

This method passes the request up the command chain by calling AllowTargetSwitch() for its SuperCommander.

Subclasses should override this function if they wish to disallow a target switch under certain circumstances.

Access Virtual, Public

Prototype

```
virtual Boolean AllowTargetSwitch(
     LCommander *inNewTarget);
```

Parameters The parameter for this method is:

`LCommander*`	`inNewTarget`	The pointer to the new target.

Return A `Boolean` indicating whether the target switch was allowed.

AttemptQuit()

Purpose This method asks all subcommanders if we can quit.

Access Virtual, Public

Prototype

```
virtual Boolean AttemptQuit(
     long inSaveOption);
```

Parameters The parameter for this method is:

`long`	`inSaveOption`	The save option to pass to AttemptQuitSelf().

Return `Boolean` indicating whether it is valid to quit, `false` if not.

AttemptQuitSelf()

Purpose This is a private method that indicates whether it is valid to quit.

Access Virtual, Protected

Prototype

```
virtual Boolean AttemptQuitSelf(
     Int32 inSaveOption );
```

Parameters The parameter for this method is:

`long`	`inSaveOption`	Unused.

Return `Boolean` indicating whether it is valid to quit, `false` if not.

BeTarget()

Purpose This method is called when the Commander is becoming the Target.

Subclasses should override this function if they wish to behave differently when they are and are not the target. At entry, the class variable sTarget points to this command.

Access Virtual, Protected

Prototype

```
virtual void BeTarget();
```

Parameters None

Return None

DontBeTarget()

Purpose This method is called when the commander will no longer be the target.

Subclasses should override this function if they wish to behave differently when they are and are not the target. At entry, the class variable sTarget points to this commander. sTarget will be changed soon afterwards to the new target.

Access Virtual, Protected

Prototype
```
virtual void DontBeTarget()
```

Parameters None

Return None

FindCommandStatus()

Purpose This method passes back the status of a command.

Subclasses must override to enable/disable and mark commands. PowerPlant uses the enabling and marking information to set the appearance of Menu items.

Access Virtual, Public

Prototype
```
virtual void FindCommandStatus(
     CommandT inCommand,
     Boolean &outEnabled,
     Boolean &outUsesMark,
     Char16 &outMark,
     Str255 outName);
```

Parameters The parameters for this method are:

`CommandT`	`inCommand`	The command passed to this method.
`Boolean&`	`outEnabled`	Tells whether the command is enabled.
`Boolean&`	`outUsesMark`	Tells whether the command uses a mark.
`Char16&`	`outMark`	The mark information for the command.
`Str255`	`outName`	The name string.

Return None

GetDefaultCommander()

Purpose This method returns the value of the sDefaultCommander data member.

Access Inline, Static, Public

Prototype
```
static LCommander* GetDefaultCommander();
```

Parameters None

Return Returns the value of sDefaultCommander.

GetLatentSub()

Purpose Find the latent subcommander of a Commander. A Commander may have one or no latent subcommander.

Access Virtual, Public

Prototype
```
virtual LCommander* GetLatentSub();
```

Parameters None

Return Returns the latent subcommander in a pointer to an LCommander.

GetSuperCommander()

Purpose Return the value of the mSuperCommander data member.

Access Inline, Public

Prototype
```
LCommander* GetSuperCommander();
```

Parameters None

Return Returns the value of mSuperCommander.

GetTarget()

Purpose Return the value of the sTarget data member.

Access Inline, Static, Public

Prototype
```
static LCommander* GetTarget();
```

Parameters None

Return Returns the value of sTarget.

GetTopCommander()

Purpose Returns the value of sTopCommander data member.

Access Inline, Static, Public

Prototype
```
static LCommander* GetTopCommander();
```

Parameters None

Return Returns the value of sTopCommander.

GetUpdateCommandStatus()

Purpose Returns the value of sUpdateCommandStatus data member.

Access Inline, Static, Public

Prototype
```
static Boolean GetUpdateCommandStatus()
```

Parameters None

Return Returns the value of sUpdateCommandStatus.

HandleKeyPress()

Purpose This method processes key presses.

Access Virtual, Public

Prototype
```
virtual Boolean HandleKeyPress(
     const EventRecord &inKeyEvent );
```

Parameters This method has the following parameter:

`const EventRecord&`	`inKeyEvent`	The key press event.

Return Boolean of `true` if the key press was handled, else `false`.

InitCommander()

Purpose Private function for initializing data members from the constructors.

Access Private

Prototype
```
void InitCommander( LCommander *inSuper );
```

Parameters This method has the following parameter:

`LCommander*`	`inSuper`	Pointer to the SuperCommander.

Return None

IsOnDuty()

Purpose Indicate whether this commander is on duty.

Access Public

Prototype
```
Boolean IsOnDuty() const;
```

Parameters None

Return Boolean of `true` if the command is on duty, else `false`.

IsSyntheticCommand()

Purpose Indicate whether a command is synthetic. If so, pass back the associated Menu ID and item number. If not synthetic, `outMenuID` and `outMenuItem` are undefined.

Access Static, Public

Prototype
```
static Boolean IsSyntheticCommand(
     CommandT inCommand,
     ResIDT &outMenuID,
     Int16 &outMenuItem);
```

Parameters This method has the following parameters:

`CommandT`	`inCommand`	Command passed to this method.
`ResIDT&`	`outMenuID`	The menu resource ID.
`Int16&`	`outMenuItem`	The menu item number.

Return Boolean of `true` if the command is synthetic, else `false`.

Remarks A synthetic command number has the Menu ID in the high 16 bits and the item number in the low 16 bits, with the result negatated.

syntheticCmd = - (MenuID << 16) - ItemNumber

A synthetic command is the negative of the value returned by the Toolbox traps `MenuSelect()` and `MenuKey()`.

The LMenu and LMenuBar classes return synthetic command numbers for menu items whose actual command number is `cmd_UseMenuItem`. You should use synthetic command numbers when the menu choice depends on the runtime name of the menu item. For example, an item in a Font menu.

IsTarget()

Purpose Return whether this Commander is the target

Access Public

Prototype
```
Boolean IsTarget() const;
```

Parameters None

Return Return `true` if the Commander is the target, else `false`.

ObeyCommand()

Purpose Issue a command to a Commander.

Subclasses must override this method in order to respond to commands.

Access Virtual, Public

Prototype

```
virtual Boolean ObeyCommand(
     CommandT inCommand,
     void *ioParam );
```

Parameters This method has the following parameters:

`CommandT`	`inCommand`	Command passed to this method.
`void*`	`ioParam`	The data block to accompany the command.

Return Returns whether the command was handled (`true`) or not (`false`).

PostAction()

Purpose If an attachment doesn't intercept the `msg_PostAction`, this function will post the action up to the supercommander of this object. The default NULL parameter will effectively clear or "commit" the last semantic action.

If an attachment didn't process the posting and there's no supercommander, this function will attempt to Redo, Finalize, and delete the action.

Access Virtual, Public

Prototype

```
virtual void PostAction( LAction *inAction );
```

Parameters This method has the following parameter:

`LAction*`	`inAction`	The action to post.

Return None

PostAnAction()

Purpose Post the action to the present target. This is a static member function that can be used by anything needing to post an action.

If there is no present target, function will attempt to Redo, Finalize, and delete the action.

Access Static, Public

Prototype

```
static void PostAnAction( LAction *inAction );
```

Parameters This method has the following parameter:

`LAction*`	`inAction`	The action to post.

Return None

ProcessCommand()

Purpose Issue a command to a Commander.

This function lets Attachments handle the Command before calling the normal ObeyCommand() function.

Access Virtual, Public

Prototype

```
virtual Boolean ProcessCommand(
     CommandT inCommand,
     void *ioParam );
```

Parameters This method has the following parameters:

`CommandT`	`inCommand`	Command passed to this method.
`void*`	`ioParam`	The data block to accompany the command.

Return Returns whether the command was handled (`true`) or not (`false`).

ProcessCommandStatus()

Purpose Pass back the status of a command. This method lets Attachments set the command status before calling the normal FindCommandStatus() method.

Access Virtual, Public

Prototype

```
virtual void ProcessCommandStatus(
     CommandT inCommand,
     Boolean &outEnabled,
     Boolean &outUsesMark,
     Char16 &outMark,
     Str255 outName );
```

Parameters The parameters for this method are:

`CommandT`	`inCommand`	Command passed to this method.
`Boolean&`	`outEnabled`	Tells whether the command is enabled.
`Boolean&`	`outUsesMark`	Tells whether the command uses a mark.
`Char16&`	`outMark`	The mark information for the command.
`Str255`	`outName`	The name string.

Return None

ProcessKeyPress()

Purpose Processes keystrokes.

Access Virtual, Public

Prototype

```
virtual Boolean ProcessKeyPress(
     const EventRecord &inKeyEvent );
```

Parameters This method has the following parameter:

`const EventRecord&`	`inKeyEvent`	The key press event.

Return Returns `true` if handled, else `false`.

PutChainOnDuty()

Purpose Put on duty a chain of Commanders. This is a wrapper function that guarantees that a Commander can't be put on duty until all its Superiors are on duty.

Access Protected

Prototype
```
void PutChainOnDuty();
```

Parameters None

Return None

PutOnDuty()

Purpose Called when a Commander is going on duty. Subclasses should override this function if they wish to behave differently when on duty than when off duty.

Access Virtual, Protected

Prototype
```
virtual void PutOnDuty();
```

Parameters None

Return None

RemoveSubCommander()

Purpose This method removes a Subcommander.

Access Virtual, Protected

Prototype
```
virtual void RemoveSubCommander(LCommander *inSub);
```

Parameters This method has the following parameter:

`LCommander*`	`inSub`	The pointer to the commander to remove.

Return None

RestoreTarget()

Purpose Set target to the Commander which was the target when this Commander was last on duty.

Access Virtual, Public

Prototype
```
virtual void RestoreTarget();
```

Parameters None

Return None

SetDefaultCommander()

Purpose Sets the value of the sDefaultCommander data member.

Access Inline, Static, Public

Prototype
```
static void SetDefaultCommander(
   LCommander *inCommander );
```

Parameters This method has the following parameter:

`LCommander*`	`inCommander`	The pointer to the commander to set.

Return None

SetLatentSub()

Purpose Specify the Subcommander that will be put on duty when this Commander is put on duty.

This method does nothing if this Commander is already on duty. `inSub` may be `nil`, in which case this Commander will have no Latent Subcommander. This will also be the case if `inSub` is not a Subordinate of this Commander (which raises a Signal).

Access Virtual, Public

Prototype

```
virtual void SetLatentSub( LCommander *inSub );
```

Parameters This method has the following parameter:

`LCommander*`	`inSub`	The pointer to the commander to remove.

Return None

SetSuperCommander()

Purpose Command chain maintenance method.

Access Virtual, Public

Prototype

```
virtual void SetSuperCommander(LCommander
    *inSuper);
```

Parameters This method has the following parameter:

`LCommander*`	`inSuper`	The pointer to the Supercommander to set.

Return None

SetTarget()

Purpose Set the Target class variable. Called internally.

Access Static, Protected

Prototype

```
static void SetTarget( LCommander *inNewTarget );
```

Parameters This method has the following parameter:

`LCommander*`	`inNewTarget`	The pointer to the Commander to set as the Target.

Return None

SetUpdateCommandStatus()

Purpose This method sets the value of the sUpdateCommandStatus data member.

Access Static, Public

Prototype

```
static void SetUpdateCommandStatus(
      Boolean inDirty );
```

Parameters This method has the following parameter:

`Boolean`	`inDirty`	The value indicating whether to mark as dirty or not.

Return None

SwitchTarget()

Purpose Try to change Target and return whether the specified Commander did indeed become the Target.

A Target switch can fail if some superior of the old and new Target disallows the switch. For example, this could happen when performing data entry validation.

Access Static, Public

Prototype

```
static Boolean SwitchTarget(
     LCommander *inNewTarget );
```

Parameters This method has the following parameter:

`LCommander*`	`inNewTarget`	The pointer to the Commander to set as the Target.

Return `Boolean` indicating whether the switch was allowed (`true`) or not (`false`).

TakeChainOffDuty()

Purpose Take a chain of Commanders off duty.

This is a wrapper function that traverses a command chain from the Commander that firsts receives this message up to the specified Commander. The `inUpToCmdr` is not taken off duty.

It is an error if `inUpToCmdr` is not a Superior of the Commander that first receives this message. Note that this includes the case where `inUpToCmdr` is the Commander that first receives this message (since a Commander is not a Superior of itself). `nil` is a valid value for `inUpToCmdr`, since the Commander at the top of a chain of command has a `nil` SuperCommander.

Access Protected

Prototype

```
void TakeChainOffDuty(const LCommander
     *inUpToCmdr);
```

Parameters This method has the following parameter:

`LCommander*`	`inUpToCmdr`	The pointer to the Commander to take off duty.

Return None

TakeOffDuty()

Purpose A Commander is going off duty. Subclasses should override this method if they wish to behave differently when on duty than when off duty.

Access Protected

Prototype
```
virtual void TakeOffDuty();
```

Parameters None

Return None

sTopCommander

Purpose The top Commander pointer.

Access Protected

Prototype
```
static LCommander *sTopCommander;
```

sTarget

Purpose The Target pointer.

Access Protected

Prototype
```
static LCommander *sTarget;
```

sDefaultCommander

Purpose The default Commander pointer.

Access Protected

Prototype
```
static LCommander *sDefaultCommander;
```

sUpdateCommandStatus

Purpose Indicates whether command status should be updated.

Access Protected

Prototype
```
static Boolean sUpdateCommandStatus;
```

mSuperCommander

Purpose The pointer to the Supercommander.

Access Protected

Prototype
```
LCommander *mSuperCommander;
```

mSubCommanders

Purpose An array of pointers to Subcommanders.

Access Protected

Prototype
```
TArray<LCommander*> mSubCommanders;
```

mOnDuty

Purpose Indicates whether the Commander is on duty or not.

Access Protected

Prototype `ETriState mOnDuty;`

LCommanderPane

Overview LCommanderPane is a PowerPlant class is used for creating Attachments to facilitate drawing, printing, clicking, and command handling.

Methods The methods in this class are:

LCommanderPane() ~LCommanderPane()

Data Members There are no data members in this class.

Operation This object is a subclass of LCommander and LPane. An object of this class does nothing by itself, but becomes powerful when combined with Attachments which control all drawing, printing, clicking, and command handling.

Source files (Utility Classes)

`UAttachments.h`

`UAttachments.cp`

See also LAttachment

LCommander

LPane

LCommanderPane()

Purpose The constructors create the object using the passed-in parameters.

Access Public

Prototype

```
LCommanderPane( SPaneInfo &inPaneInfo,
     LCommander *inSuper );
LCommanderPane( LStream *inStream );
```

Parameters The parameters for these constructors are:

`SPaneInfo&`	`inPaneInfo`	A reference to the Pane that works in conjunction with LCommanderPane.
`LCommander*`	`inSuper`	A pointer to the SuperView that works in conjunction with LCommanderPane.
`LStream*`	`inStream`	A pointer to a stream object that contains the information to create the LCommanderPane object.

~LCommanderPane()

Purpose The destructor destroys the object.

Access Virtual, Public

Prototype

```
virtual ~LCommanderPane();
```

LComparator

Description LComparator is a PowerPlant class that is used for comparing objects. Comparators are objects that know how to compare to other objects or structures.

Methods The methods in this class are:

LComparator()	~LComparator()
Clone()	Compare()
CompareToKey()	GetComparator()
IsEqualTo()	IsEqualToKey()

Data Members The data members in this class are:

sComparator

Operation Subclasses will need to implement the Compare() method. The compare Compare() result should be one of the following:

- < 0 — object/data item 1 is less than object/data item 2
- 0 — object/data item 1 is equal to object/data item 2
- > 0 — object/data item 1 is greater than object/data item 2

Source files (Array Classes)

`LComparator.h`

`LComparator.cp`

LComparator()

Purpose The constructor is a placeholder, it doesn't do anything for this class.

Access Public

Prototype

```
LComparator();
```

Parameters None

~LComparator()

Purpose The destructor is a placeholder, it doesn't do anything for this class.

Access Virtual, Public

Prototype
```
~LComparator();
```

Clone()

Purpose This method simply creates this object by calling `new`.

Access Virtual, Public

Prototype
```
virtual LComparator* Clone();
```

Parameters None

Return A pointer to a new comparator object.

Compare()

Purpose This method takes two pointers to data blocks, and their sizes, and determines what the result of the comparison should be.

Access Virtual, Public

Prototype
```
virtual Int32 Compare(
      const void* inItemOne,
      const void* inItemTwo,
      Uint32 inSizeOne,
      Uint32 inSizeTwo ) const;
```

Parameters This method has the following parameters:

`const void*`	`inItemOne`	A data pointer to the first item.
`const void*`	`inItemTwo`	A data pointer to the second item.
`Uint32`	`inSizeOne`	The size of the first item.
`Uint32`	`inSizeTwo`	The size of the second item.

Return Int32 that indicates one of the following:

- < 0 — object/data item 1 is less than object/data item 2
- 0 — object/data item 1 is equal to object/data item 2
- > 0 — object/data item 1 is greater than object/data item 2

CompareToKey()

Purpose This method is not yet implemented.

Access Virtual, Public

Prototype

```
virtual Int32 CompareToKey(
      const void* inItem,
      Uint32 inSize,
      const void* inKey) const;
```

Parameters This method has the following parameters:

`const void*`	`inItem`	The pointer to the first item.
`Uint32`	`inSize`	The size of the first item.
`const void*`	`inKey`	The key to compare with.

Return Int32

GetComparator()

Purpose This method is an accessor for sComparator. If sComparator does not yet exist, the object is created before returning from this method.

Access Static, Public

Prototype

```
static LComparator* GetComparator();
```

Parameters None

Return A pointer to sComparator.

IsEqualTo()

Purpose This method detects whether two objects are equal.

Access Virtual, Public

Prototype

```
virtual Boolean IsEqualTo(
      const void* inItemOne,
      const void* inItemTwo,
      Uint32 inSizeOne,
      Uint32 inSizeTwo) const;
```

Parameters This method has the following parameters:

`const void*`	`inItemOne`	A data pointer to the first item.
`const void*`	`inItemTwo`	A data pointer to the second item.
`Uint32`	`inSizeOne`	The size of the first item.
`Uint32`	`inSizeTwo`	The size of the second item.

Return `Boolean` indicating `true` if the items are equal, `false` otherwise.

IsEqualToKey()

Purpose This method compares an item to the key to see if they are identical.

Access Virtual, Public

Prototype

```
virtual Boolean IsEqualToKey(
      const void* inItem,
      Uint32 inSize,
      const void* inKey) const;
```

Parameterss This method has the following parameters:

`const void*`	`inItem`	The pointer to the first item.
`Uint32`	`inSize`	The size of the first item.
`const void*`	`inKey`	The key to compare with.

Return `Boolean` indicating `true` if the items are equal, `false` otherwise.

sComparator

Purpose This data member stores the pointer to the object we use as a standard for comparisons.

Access Static, Protected

Prototype

```
static LComparator* sComparator;
```

LControl

Overview

LControl is a PowerPlant class that is used for manipulation and control of radio buttons, check boxes, pop-up menus, and other visual interface controls.

A control is an object of any class which inherits from the LControl class. All the standard Mac OS controls are provided by PowerPlant, and some additional controls as well.

Methods

The methods in this class are:

LControl()	~LControl()
BroadcastValueMessage()	ClickSelf()
FindHotSpot()	GetMaxValue()
GetMinValue()	GetValue()
GetValueMessage()	HotSpotAction()
HotSpotResult()	IncrementValue()
PointInHotSpot()	SetMaxValue()
SetMinValue()	SetValue()
SetValueMessage()	SimulateHotSpotClick()
TrackHotSpot()	

Data Members

The data members in this class are:

mValueMessage	mValue
mMinValue	mMaxValue

Operation

Controls are a large topic of discussion. In order to master the concepts and terminology of Controls within PowerPlant, a careful reading of the "Controls and Messaging" chapter in *The PowerPlant Book* is a good thing.

Source files

(Pane Classes)

`LControl.h`

`LControl.cp`

Ancestors LAttachable

LBroadcaster

LPane

LControl()

Purpose The constructors create objects from the passed-in parameters.

Access Public

Prototype

```
LControl();
LControl( const LControl &inOriginal );
LControl( const SPaneInfo &inPaneInfo,
     MessageT inValueMessage,
     Int32 inValue,
     Int32 inMinValue,
     Int32 inMaxValue );
LControl( LStream *inStream );
```

Parameters These constructors have the following parameters:

`const LControl&`	`inOriginal`	A reference to the `LControl` object you want to copy.
`const SPaneInfo&`	`inPaneInfo`	A reference to the SPaneInfo object that is the super view.
`MessageT`	`inValueMes-sage`	The message sent when the control is interacted with.
`Int32`	`inValue`	The value of the control.
`Int32`	`inMinValue`	The minimum control value.

`Int32`	`inMaxValue`	The maximum control value.
`LStream*`	`inStream`	A pointer to a stream object that contains the information to create the LControl object.

~LControl()

Purpose The destructor destroys the object.

Access Virtual, Public

Prototype

```
virtual ~LControl();
```

BroadcastValueMessage()

Purpose Broadcasts a message indicating a control change to the Listeners.

Access Virtual, Protected

Prototype

```
virtual void BroadcastValueMessage();
```

Parameters None

Return None

ClickSelf()

Purpose Tracks the mouse click and determines if a hot spot within a control was selected. If the hot spot was selected then this method responds to the mouse click by calling the HotSpotResult() method defined for the class.

Access Virtual, Protected

Prototype

```
virtual void ClickSelf(
     const SMouseDownEvent &inMouseDown );
```

Parameters This method has the following parameters:

`const` SMouseDownEvent&	`inMouseDown`	The event information for the mouse down.

Return None

FindHotSpot()

Purpose Determine which hot spot, if any, was clicked.

Access Virtual, Protected

Prototype
```
virtual Int16 FindHotSpot( Point inPoint );
```

Parameters This method has the following parameters:

`Point`	`inPoint`	Local coordinates for the Control.

Return `Int16` containing the value of the hot spot selected.

Remarks If the return value is 0 then no hot spot was selected. Valid hot spots are integer values greater than 0.

GetMaxValue()

Purpose Get the maximum value for the control, found in the mMaxValue data member.

Access Public

Prototype
```
Int32 GetMaxValue() const;
```

Parameters None

Return `Int32` containing the value in mMaxValue.

GetMinValue()

Purpose Get the minimum value for the control, found in the mMinValue data member.

Access Public

Prototype
```
Int32 GetMinValue() const;
```

Parameters None

Return `Int32` containing the value in mMinValue.

GetValue()

Purpose Return the current value of a Control.

Access Virtual, Public

Prototype
```
virtual Int32 GetValue() const;
```

Parameters None

Return `Int32` containing the value in mValue

GetValueMessage()

Purpose Returns the value of the message.

Access Public

Prototype
```
MessageT GetValueMessage() const;
```

Parameters None

Return `MessageT` containing the value in mValueMessage.

HotSpotAction()

Purpose This method allows you to take action while the hot spot is down clicked by the mouse.

Access Virtual, Protected

Prototype

```
virtual void HotSpotAction( Int16 inHotSpot,
     Boolean inCurrInside,
     Boolean inPrevInside );
```

Parameters This method has the following parameters:

`Int16`	`inHotSpot`	The hot spot number.
`Boolean`	`inCurrInside`	Parameter is true if the mouse is currently inside the hot spot.
`Boolean`	`inPrevInside`	Parameter is true if the mouse was previously in the hot spot.

Return None

Remarks Depending on whether the mouse in inside or outside the hot spot you can override to change the control's visual appearance.

While the mouse is down you can override to continuously perform some action.

HotSpotResult()

Purpose Perform an action as a result of clicking and releasing the mouse inside a hot spot.

Access Virtual, Protected

Prototype

```
virtual void HotSpotResult( Int16 inHotSpot );
```

Parameters This method has the following parameters:

`Int16`	`inHotSpot`	The hotspot number.

Return None

Remarks Subclasses need to override this function to implement behavior associated with clicking in a Control hot spot.

IncrementValue()

Purpose Increment the current value of the control.

Access Virtual, Public

Prototype

```
virtual void IncrementValue( Int32 inIncrement );
```

Parameters This method has the following parameters:

`Int32`	`inIncrement`	The amount the current control value is incremented.

Return None

Remarks The incremental change in the amount of the value can be a negative amount.

PointInHotSpot()

Purpose Determine if the mouse is in the hot spot while down-clicking the mouse.

Access Virtual, Protected

Prototype

```
virtual Boolean PointInHotSpot( Point inPoint,
     Int16 inHotSpot);
```

Parameters This method has the following parameters:

`Point`	`inPoint`	The local coordinates for the control.
`Int16`	`inHotSpot`	The hot spot number.

Return Boolean true if the point is with the hot spot, else false.

SetMaxValue()

Purpose If the value is greater than mMaxValue then set to the maximum value.

Access Virtual, Public

Prototype

```
virtual void SetMaxValue( Int32 inMaxValue );
```

Parameters This method has the following parameters:

`Int32`	`inMaxValue`	Maximum value to set for the control.

Return None

SetMinValue()

Purpose If the value is less than mMinValue then set to the minimum value.

Access Virtual, Public

Prototype

```
virtual void SetMinValue( Int32 inMinValue);
```

Parameters This method has the following parameters:

`Int32`	`inMinValue`	The minimum value to set for the control.

Return None

SetValue()

Purpose If the value has changed then set the new value and inform Listeners of the value change.

Access Virtual, Public

Prototype
```
virtual void SetValue( Int32 inValue );
```

Parameters This method has the following parameters:

`Int32`	`inValue`	The new value to set for the control.

Return None

SetValueMessage()

Purpose Sets the value of the message.

Access Public

Prototype
```
void SetValueMessage( MessageT inValueMessage);
```

Parameters This method has the following parameters:

`MessageT`	`inValueMessage`	The value to set for the message.

Return None

SimulateHotSpotClick()

Purpose This method allows you to simulate clicking the hot spot of a Control. Visual feedback is provided as if the user had clicked inside the hot spot of a Control.

HotSpotAction() is called to simulate a down of the mouse. Then HotSpotResult() is used to simulate release within a particular hot spot.

Access Virtual, Public

Prototype

```
virtual void SimulateHotSpotClick(Int16 inHotSpot);
```

Parameters This method has the following parameters:

`Int16`	`inHotSpot`	The hot spot number.

Return None

Remarks This function can be used to implement keystroke equivalents for `Control-Click` operations.

TrackHotSpot()

Purpose After down clicking this function tracks the mouse and PointInHotSpot() determines if a point is within a particular hopspot. Movement inside and outside the hotspot is tracked. HotSpotAction() is called to take action during mouse tracking.

Once the mouse is released the location of the event is recorded.

Access Virtual, Protected

Prototype

```
virtual Boolean TrackHotSpot( Int16 inHotSpot,
    Point inPoint);
```

Parameters This method has the following parameters:

`Int16`	`inHotSpot`	The hotspot number.
`Point`	`inPoint`	Local coordinates for the control.

Return Boolean true if mouse released inside the hot spot, else false.

mValueMessage

Purpose This data member is set using SetValueMessage(). The message is broadcast to all Listeners via BroadcastValueMessage().

Access Protected

Prototype
```
MessageT mValueMessage;
```

mValue

Purpose Contains the value for the control.

Access Protected

Prototype
```
Int32 mValue;
```

mMinValue

Purpose Minimum value for the control.

Access Protected

Prototype
```
Int32 mMinValue;
```

mMaxValue

Purpose Maximum value for the control.

Access Protected

Prototype
```
Int32 mMaxValue;
```

LDataArrived

Overview LDataArrived is a PowerPlant class that assists in the task of sending a message when data arrives via an asynchronous channel.

Methods The methods in this class are:

LDataArrived() ~LDataArrived()

GetDataBuffer() GetDataSize()

GetRemoteAddress()

Data Members The data members in this class are:

mDataBuffer mDataSize

mRemoteAddress mMustReleaseMemory

Source files (Networking Classes)

`UNetworkMessages.h`

`UNetworkMessages.cp`

See also LNetMessage

LRMPObject

LDataArrived()

Purpose The constructor creates the object.

Access

Prototype

```
LDataArrived(
     MessageT inMessageType,
     ResultT inResultCode,
     void* inDataBuffer,
     Uint32 inDataSize,
     LInternetAddress* inRemoteAddress,
     Boolean inMustReleaseMemory,
```

```
    LEndpoint* inEndpoint );
LDataArrived(); /* do not use */
LDataArrived(LDataArrived&); /* do not use */
```

Parameters The parameters for this constructor are:

`MessageT`	`inMessageType`	The message.
`ResultT`	`inResultCode`	The result code.
`void*`	`inDataBuffer`	The buffer.
`Uint32`	`inDataSize`	The size of the data.
`LInternetAddress*`	`inRemoteAddress`	The remote address.
`Boolean`	`inMustReleaseMemory`	Whether to release memory or not.
`LEndpoint*`	`inEndpoint`	The network endpoint.

~LDataArrived()

Purpose The destructor destroys the object.

Access Virtual, Public

Prototype

```
virtual ~LDataArrived();
```

GetDataBuffer()

Purpose Retrieve the value of the mDataBuffer data member.

Access Public

Prototype

```
void* GetDataBuffer() const;
```

Parameters None

Return Returns the value of mDataBuffer.

GetDataSize()

Purpose Retrieve the size of the data, held in the mDataSize data member.

Access Public

Prototype
```
Uint32 GetDataSize() const;
```

Parameters None

Return The value of mDataSize.

GetRemoteAddress()

Purpose Retrieve the value of the mRemoteAddress data member.

Access Public

Prototype
```
LInternetAddress* GetRemoteAddress() const;
```

Parameters None

Return The address, held in mRemoteAddress.

mDataBuffer

Purpose The data buffer.

Access Protected

Prototype
```
void* mDataBuffer;
```

mDataSize

Purpose The size of the data.

Access Protected

Prototype
```
Uint32 mDataSize;
```

mRemoteAddress

Purpose The address.

Access Protected

Prototype
```
LInternetAddress* mRemoteAddress;
```

mMustReleaseMemory

Purpose Whether to release memory or not.

Access Protected

Prototype
```
Boolean mMustReleaseMemory;
```

LDatabase

Overview LDatabase is a PowerPlant abstract base class for database access.

Methods The methods in this class are:

LDatabase()	~LDatabase()
CloseDatabase()	DecodeString()
EncodeString()	GetDatabaseState()
NewDatabaseCursor()	OpenDatabase()
SetAccessTimeout()	SetDatabaseLogin()

Data Members The data members in this class are:

mDatabaseState	mUserName[33]
mPassword[33]	

Operation This class is an abstract base class that you should inherit from.

Source files (_Advanced Classes:Database Access)

`UDatabase.h`

`UDatabase.cp`

See Also LODBCApplication

LDatabaseCursor

LDatabase()

Purpose The default constructor initializes the object.

Access Public

Prototype

```
LDatabase();
```

Parameters None

~LDatabase()

Purpose The destructor destroys the object.

Access Public, Virtual

Prototype
```
virtual ~LDatabase();
```

Parameters None

CloseDatabase()

Purpose Closes the database. Implement this in your subclass.

Access Virtual, Public

Prototype
```
virtual void CloseDatabase();
```

Parameters None

Return None

DecodeString()

Purpose This function should be overloaded to do any application specific decoding of strings from the RDBMS. This should be implemented by your subclass.

Access Virtual, Public

Prototype
```
virtual void DecodeString(i8* ioString, i32
     inStringSize );
```

Parameters `ioString` is a null terminated string, `inStringSize` should be the max size of the string.

Return None

EncodeString()

Purpose This function should be overloaded to do any application specific encoding of strings from the RDBMS. This should be implemented by your subclass.

Access Virtual, Public

Prototype
```
virtual void EncodeString(i8* ioString, i32
     inStringSize );
```

Parameters `ioString` is a null terminated string, `inStringSize` should be the max size of the string.

Return None

GetDatabaseState()

Purpose Return a pointer to the database, held in mDatabaseState.

Access Virtual, Public

Prototype
```
virtual LDatabase *GetDatabaseState();
```

Parameters None

Return A pointer to the database.

NewDatabaseCursor()

Purpose Create a database cursor. This should be implemented by your sub-class.

Access Virtual, Public, Inline

Prototype
```
virtual LDatabaseCursor*NewDatabaseCursor(){
```

```
return NULL;}
```

Parameters None

Return A pointer to the database cursor.

OpenDatabase()

Purpose Opens the database. Implement this in your subclass.

Access Virtual, Public

Prototype

```
virtual void OpenDatabase();
```

Parameters None

Return None

SetAccessTimeout()

Purpose Set a default timeout for database transactions. This should be implemented in your subclass.

Access Virtual, Public

Prototype

```
virtual void SetAccessTimeout(ui32 msec);
```

Parameters The timeout in milliseconds.

Return None

SetDatabaseLogin()

Purpose Set the username and password to be used to connect to the database.

Access Virtual, Public

Prototype
```
SetDatabaseLogin(
     const i8*inUserName, const i8 *inPassword);
SetDatabaseLogin(ConstStr32Param inUserName,
     ConstStr32Param inPassword);
```

Parameters The username and password.

Return None

mUserName[33]

Purpose The username.

Access Protected

Prototype `i8` mUserName[33];

mPassword[33]

Purpose The password.

Access Protected

Prototype `i8` mPassword[33];

mDatabaseState

Purpose This data member stores the database state.

Access Protected

Prototype
```
LODBCDatabase *mDatabase;
```

LDatabaseView

Overview

LDatabaseView is a PowerPlant class that manages database access for its subviews. This is particulary useful for displaying and editing a database record where each field of the record has a corresponding LEditField, LStdCheckBox, etc.

Methods

The methods in this class are:

LDatabaseView()	~LDatabaseView()
BuildDelete()	BuildInsert()
BuildSelect()	BuildUpdate()
ClearSubPanes()	DeleteRecord()
DoSelect()	FinishCreateSelf()
GetDatabaseCursor()	InsertRecord()
IsInNewMode()	IsRecordDirty()
ListenToMessage()	PopulateSubPanes()
SetCurrentRecord()	UpdateRecord()
WasChanged()	

Data Members

The data members in this class are:

mNewMode	mDirtyFlag
mDatabaseCursor	

Operation

To use this class you will need to create a subclass and override the following:

```
virtual void BuildSelect(LStream *inStream);
virtual void BuildInsert(LStream *inStream);
virtual void BuildUpdate(LStream *inStream);
virtual void BuildDelete(LStream *inStream);
virtual void PopulateSubPanes();
virtual void ClearSubPanes();
virtual Boolean IsRecordDirty();
```

BuildSelect, BuildInsert, BuildUpdate, BuildDelete all take an LStream as input. You should write your SQL string to the LStream. It will help to have a basicunderstanding of SQL which should take most C++ developers about 2-3 hours. The best way to learn is use an SQL query tool to execute some SQL statements.

Things don't get complicated until you start dealing with referential integrity, compound indexes, stored procedures, etc. If you don't know what those things are you probably won't need them for your application.

PopulateSubPanes is where you need to copy data from the LDatabaseCursor to your edit panes. ClearSubPanes is called to clear all of your edit panes and IsRecordDirty should compare the current values of fields to the current cursor values and return true if the user has made modification. You could make your edit panes send the 'dirt' message to the LDatabaseView in which case you would not need to override IsRecordDirty.

Source files (_Advanced Classes:Database Access)

`LDatabaseView.h`

`LDatabaseView.cp`

Ancestors LListener

LView

See Also LDatabaseCursor

LDatabaseView()

Purpose The default constructor initializes the object. There is also a constructor for stream input with LStream.

Access Public

Prototype

```
LDatabaseView();
LDatabaseView(LStream *inStream);
```

Parameters Optional, you can pass a pointer to an LStream object.

~LDatabaseView()

Purpose The destructor destroys the object.

Access Public, Virtual

Prototype
```
virtual ~LDatabaseView();
```

Parameters None

BuildDelete()

Purpose Writes the SQL delete to the byte stream.

Access Virtual, Public

Prototype
```
virtual void BuildDelete(LStream *inStream);
```

Parameters A pointer to an LStream object.

Return None

BuildInsert()

Purpose Writes the SQL insert to the byte stream.

Access Virtual, Public

Prototype
```
virtual void BuildInsert(LStream *inStream);
```

Parameters A pointer to an LStream object.

Return None

BuildSelect()

Purpose Writes the SQL select to the byte stream.

Access Virtual, Public

Prototype
```
virtual void BuildSelect(LStream *inStream);
```

Parameters A pointer to an LStream object.

Return None

BuildUpdate()

Purpose Writes the SQL insert to the byte stream

Access Virtual, Public

Prototype
```
virtual void BuildUpdate(LStream *inStream);
```

Parameters A pointer to an LStream object.

Return None

ClearSubPanes()

Purpose Write default values into all subpanes.

Access Virtual, Public

Prototype
```
virtual void ClearSubPanes();
```

Parameters None

Return None

DeleteRecord()

Purpose Calls BuildDelete() to construct the SQL string to perform the update

Access Virtual, Public

Prototype
```
virtual void DeleteRecord();
```

Parameters None

Return None

DoSelect()

Purpose Calls BuildSelect() to build the SQL string, then sets the record to the first record.

Access Virtual, Public

Prototype
```
virtual void DoSelect();
```

Parameters None

Return None

FinishCreateSelf()

Purpose Adds default listeners.

Access Virtual, Public

Prototype
```
virtual void FinishCreateSelf();
```

Parameters None

Return None

GetDatabaseCursor()

Purpose Return the value of mDatabaseCursor, the current LDatabaseCursor.

Access Virtual, Public

Prototype
```
virtual LDatabaseCursor *GetDatabaseCursor();
```

Parameters None

Return A pointer to a value of type `LDatabaseCursor.`

InsertRecord()

Purpose Calls BuildInsert() to construct the SQL string to perform the insert.

Access Virtual, Public

Prototype
```
virtual void InsertRecord();
```

Parameters None

Return None

IsInNewMode()

Purpose Check the mNewMode flag.

Access Virtual, Public

Prototype
```
virtual Boolean IsInNewMode();
```

Parameters None

Return Boolean with the value of mNewMode.

IsRecordDirty()

Purpose Checks the mDirtyFlag.

Access Virtual, Public

Prototype
```
virtual Boolean IsRecordDirty();
```

Parameters None

Return Boolean with the value of mDirtyFlag.

IsInNewMode()

Purpose Check the mNewMode flag.

Access Virtual, Public

Prototype
```
virtual Boolean IsInNewMode();
```

Parameters None

Return Boolean with the value of mNewMode.

ListenToMessage()

Purpose Handles default messages, this is an override of ListenToMessage() in LListener.

PopulateSubPanes()

Purpose Copy data from the current record to any dependent views.

Access Virtual, Public

Prototype
```
virtual void PopulateSubPanes();
```

Parameters None

Return None

SetCurrentRecord()

Purpose Sets the database cursor.

Access Virtual, Public

Prototype
```
virtual void SetCurrentRecord(Int32 inRecordIndex);
```

Parameters The index to assign.

Return None

UpdateRecord()

Purpose Calls BuildUpdate to construct the SQL string to perform the update.

Access Virtual, Public

Prototype
```
virtual void UpdateRecord();
```

Parameters None.

Return None

WasChanged()

Purpose This method should be overriden to notify dependents of a change.

Access Virtual, Public

Prototype
```
virtual Boolean WasChanged();
```

Parameters None

Return Boolean of true if a change occurred, else false.

mNewMode

Purpose Set if this is a new record

Access Protected

Prototype
```
Boolean mNewMode;
```

mDirtyFlag

Purpose Set if the user has modified an editable pane

Access Protected

Prototype
```
Boolean mDirtyFlag;
```

mDatabaseCursor

Purpose Reference to a database cursor

Access Protected

Prototype
```
LDatabaseCursor *mDatabaseCursor;
```

LDBApplication

Overview LDBApplication is a PowerPlant class that wraps database access.

Methods The methods in this class are:

LDBApplication() ~LDBApplication()

OpenDatabase() CloseDatabase()

GetDatabase()

Data Members The data members in this class are:

mDatabase mUserName[32]

mPassword[32]

Operation This class sets a default user name and password for access of the database, and allows you to open, close, and obtain a pointer to the database.

Source files (_Advanced Classes:Database Access)

`LDBApplication.h`

`LDBApplication.cp`

Ancestors LApplication

See Also LODBCApplication

LDBApplication()

Purpose The default constructor initializes the object.

Access Public

Prototype

```
LDBApplication();
```

Parameters None

~LDBApplication()

Purpose The destructor destroys the object.

Access Public, Virtual

Prototype `virtual ~LDBApplication();`

Parameters None

OpenDatabase()

Purpose Opens the database.

Access Virtual, Public

Prototype `virtual void OpenDatabase();`

Parameters None

Return None

CloseDatabase()

Purpose Closes the database.

Access Virtual, Public

Prototype `virtual void Close Database();`

Parameters None

Return None

GetDatabase()

Purpose Return a pointer to the database, held in mDatabase.

Access Virtual, Public

Prototype
```
virtual LDatabase *GetDatabase();
```

Parameters None

Return A pointer to the database.

mUserName[32]

Purpose Default user name set to “dtfadm”

Access Protected

Prototype
```
Int8 mUserName[32];
```

mPassword[32]

Purpose Default password set to “dtfadm”

Access Protected

Prototype
```
Int8 mPassword[32];
```

mDatabase

Purpose This data member stores the database pointer.

Access Protected

Prototype
```
LODBCDatabase *mDatabase;
```

LDBExchangeView

Overview LDBExchangeView is a PowerPlant class that wraps database access. It manages database access for its subviews.

Methods The methods in this class are:

LDBExchangeView()	~LDBExchangeView()
BuildDelete()	BuildInsert()
BuildSelect()	BuildUpdate()
ClearSubPanes()	GetDatabaseCursor()
IsRecordDirty()	PopulateSubPanes()

Data Members The data members in this class are:

mDBXResID	mAssociationResID

Operation This class sets a default user name and password for access of the database, and allows you to open, close, and obtain a pointer to the database.

Source files (_Advanced Classes:Database Access)

`LDBExchangeView.h`

`LDBExchangeView.cp`

Ancestors LDatabaseView

See Also LStream

LDBExchangeView()

Purpose The default constructor initializes the object.

Access Public

Prototypes
```
LDBExchangeView();
LDBExchangeView(LStream*inStream);
```

Parameters A pointer to an LStream object is optional.

~LDBExchangeView()

Purpose The destructor destroys the object.

Access Public, Virtual

Prototype
```
virtual ~LDBExchangeView();
```

Parameters None

BuildDelete()

Purpose This is an override of BuildDelete() in LDatabaseView.

BuildInsert()

Purpose This is an override of BuildInsert() in LDatabaseView.

BuildSelect()

Purpose This is an override of BuildSelect() in LDatabaseView.

BuildUpdate()

Purpose This is an override of BuildUpdate() in LDatabaseView.

ClearSubPanes()

Purpose This is an override of ClearSubPanes() in LDatabaseView.

GetDatabaseCursor()

Purpose This is an override of GetDatabaseCursor() in LDatabaseView.

IsRecordDirty()

Purpose This is an override of IsRecordDirty() in LDatabaseView.

PopulateSubPanes()

Purpose This is an override of PopulateSubPanes() in LDatabaseView.

mDBXResID

Purpose The database identifier (ID of `DBiL` Resource).

Access Protected

Prototype ResIDT mDBXResID;

mAssociationResID

Purpose ID of `DBvL` Resource

Access Protected

Prototype ResIDT mAssociationResID;

LDBPopup

Overview LDBPopup is a PowerPlant helper class for database access. It Manages database access for its subviews.

Methods The methods in this class are:

LDBPopup() ~LDBPopup()

DoSelect() FinishCreateSelf()

GetSelectedValue()

Data Members The data members in this class are:

mSQLString mDisplayColumn

mValueColumn mDatabaseCursor

Operation This class sets a default user name and password for access of the database, and allows you to open, close, and obtain a pointer to the database.

Source files (_Advanced Classes:Database Access)

`LDBPopup.h`

`LDBPopup.cp`

Ancestors LStdPopupMenu

See Also LDatabaseCursor

LDatabaseView

LDBPopup()

Purpose The default constructor initializes the object.

Access Public

Prototype `LDBPopup();`

Parameters None

~LDBPopup()

Purpose The destructor destroys the object.

Access Public, Virtual

Prototype `virtual ~LDBPopup();`

Parameters None

DoSelect()

Purpose Adds menu items based on results from the SQL query.

Access Virtual, Public

Prototype `virtual void DoSelect();`

Parameters None

Return None

FinishCreateSelf()

Purpose Calls DoSelect().

Access Virtual, Public

Prototype `virtual void FinishCreateSelf();`

Parameters None

Return None

GetSelectedValue()

Purpose Get the value selected.

Access Virtual, Public

Prototype `virtual void GetSelectedValue(Str255 outValue);`

Parameters None

Return A pointer to the database.

mSQLString

Purpose The SQL to issue to obtain the menu items.

Access Protected

Prototype Str255 mSQLString;

mDisplayColumn

Purpose The column of the query results to display in the menu items.

Access Protected

Prototype Int32 mDisplayColumn;

mValueColumn

Purpose The column of the query results to return in GetSelectedValue.

Access Protected

Prototype Int32 mValueColumn;

mDatabaseCursor

Purpose Pointer to a database cursor.

Access Protected

Prototype

```
LDatabaseCursor *mDatabaseCursor;
```

LDBTable

Overview

LDBTable is a PowerPlant class that displays the results of an SQL query in a table.

Methods

The methods in this class are:

LDBTable()	~LDBTable()
AdjustTable()	DoSelect()
DrawCell()	DrawSelf()
FinishCreateSelf()	GetTextTraits()
SetTextTraits()	

Data Members

The data members in this class are:

mQueryString	mTextTraitsID
mDatabaseCursor	

Operation

One nice thing about this class is that data for the entire table is not copied. Results are obtained from the database cursor on an as needed basis. The SQL query string is entered via Constructor. The example provides the Constructor templates to allow you to edit LDBTables.

LDBTable is nearly identical to LDBTableView but LDBTableView is a subclass of LTableView which allows you to set column width but was an "in progress" class when creating the database classes.

Source files

(_Advanced Classes:Database Access)

`LDBTable.h`

`LDBTable.cp`

Ancestors

LTable

See Also

LDBTableView

LDBTable()

Purpose The default constructor initializes the object.

Access Public

Prototype
```
LDBTable();
LDBTable(LStream *inStream);
```

Parameters There is an optional parameter that is the stream.

~LDBTable()

Purpose The destructor destroys the object.

Access Public, Virtual

Prototype
```
virtual ~LDBTable();
```

Parameters None

AdjustTable()

Purpose Opens the database.

Access Virtual, Public

Prototype
```
virtual void AdjustTable();
```

Parameters None

Return None

DoSelect()

Purpose Get data.

Access Virtual, Public

Prototype `virtual void DoSelect();`

Parameters None

Return None

DrawCell()

Purpose Draws the cell.

Access Virtual, Public

Prototype `virtual voidDrawCell(const TableCellT &inCell);`

Parameters The cell reference.

Return None

DrawSelf()

Purpose Draws the cell.

Access Virtual, Public

Prototype `virtual void DrawSelf();`

Parameters None

Return None

FinishCreateSelf()

Purpose Calls DrawSelf().

Access Virtual, Public

Prototype `virtual void FinishCreateSelf();`

Parameters None

Return None

GetTextTraits()

Purpose Returns the value of mTextTraitsID.

Access Inline, Public

Prototype
```
ResIDT GetTextTraitsID() const
   { return mTextTraitsID; }
```

Parameters None

Return The resource id for the text traits resource.

SetTextTraits()

Purpose Sets the value of mTextTraitsID.

Access Inline, Public

Prototype
```
void SetTextTraitsID( ResIDTinTxtrID)
     { mTextTraitsID = inTxtrID; }
```

Parameters None

Return The resource id for the text traits resource.

mQueryString

Purpose Query string.

Access Protected

Prototype
```
Str255 mQueryString;
```

mTextTraitsID

Purpose The resource id for the text traits resource.

Access Protected

Prototype

```
ResIDT mTextTraitsID;
```

mDatabaseCursor

Purpose This database cursor.

Access Protected

Prototype

```
LDatabaseCursor *mDatabaseCursor;
```

LDBTableView

Overview

LDBTableView is a PowerPlant class that Displays the results of an SQL Query in a table.

Methods

The methods in this class are:

LDBTableView()	~LDBTableView()
AdjustTable()	DoSelect()
DrawCell()	DrawSelf()
FinishCreateSelf()	GetTextTraits()
SetTextTraits()	

Data Members

The data members in this class are:

mQueryString	mTextTraitsID
mDatabaseCursor	

Operation

One nice thing about this class is that data for the entire table is not copied. Results are obtained from the database cursor on an as needed basis. The SQL query string is entered via Constructor. The example provides the Constructor templates to allow you to edit LDBTables.

LDBTable is nearly identical to LDBTableView but LDBTableView is a subclass of LTableView which allows you to set column width.

Source files

(_Advanced Classes:Database Access)

`LDBTableView.h`

`LDBTableView.cp`

Ancestors

LTable

See Also

LDBTable

LStream

LDBTableView()

Purpose The default constructor initializes the object.

Access Public

Prototype
```
LDBTableView();
LDBTableView(LStream *inStream);
```

Parameters There is an optional parameter that is the stream.

~LDBTableView()

Purpose The destructor destroys the object.

Access Public, Virtual

Prototype
```
virtual ~LDBTableView();
```

Parameters None

AdjustTable()

Purpose Opens the database.

Access Virtual, Public

Prototype
```
virtual void AdjustTable();
```

Parameters None

Return None

DoSelect()

Purpose Get data.

Access Virtual, Public

Prototype
```
virtual void DoSelect();
```

Parameters None

Return None

DrawCell()

Purpose Draws the cell.

Access Virtual, Public

Prototype
```
virtual voidDrawCell(const TableCellT &inCell);
```

Parameters The cell reference.

Return None

DrawSelf()

Purpose Draws the cell.

Access Virtual, Public

Prototype
```
virtual void DrawSelf();
```

Parameters None

Return None

FinishCreateSelf()

Purpose Calls DrawSelf().

Access Virtual, Public

Prototype
```
virtual void FinishCreateSelf();
```

Parameters None

Return None

GetTextTraits()

Purpose Returns the value of mTextTraitsID.

Access Inline, Public

Prototype
```
ResIDT GetTextTraitsID() const
   { return mTextTraitsID; }
```

Parameters None

Return The resource id for the text traits resource.

SetTextTraits()

Purpose Sets the value of mTextTraitsID.

Access Inline, Public

Prototype
```
void SetTextTraitsID( ResIDTinTxtrID)
     { mTextTraitsID = inTxtrID; }
```

Parameters None

Return The resource id for the text traits resource.

mQueryString

Purpose Query string.

Access Protected

Prototype

```
Str255 mQueryString;
```

mTextTraitsID

Purpose The resource id for the text traits resource.

Access Protected

Prototype

```
ResIDT mTextTraitsID;
```

mDatabaseCursor

Purpose This database cursor.

Access Protected

Prototype

```
LDatabaseCursor *mDatabaseCursor;
```

LDefaultOutline

Overview LDefaultOutline is a PowerPlant class that is used for drawing an outline around the default button in a dialog.

Methods The methods in this class are:

LDefaultOutline() DrawSelf()

Data Members There are no data members in this class.

Operation You typically won't create an object of this class yourself. PowerPlant makes one for you when you create a dialog box and specify a default button.

Source files (Pane Classes)

`LLStdControl.h`

`LLStdControl.cp`

Ancestors LAttachable

LPane

LDefaultOutline()

Purpose This method onstructs an outline for the specified host Pane.

Access Public

Prototype

```
LDefaultOutline( LPane* inHostPane );
```

Parameters The parameter for this constructor is:

`LPane*`	`inHostPane`	This is a pointer to the host pane object.

DrawSelf()

Purpose This method draws the outline. This is an override of DrawSelf() in LPane.

LDialogBox

Overview

LDialogBox is a PowerPlant class that is used for creating dialog boxes in your program.

Methods

The methods in this class are:

LDialogBox()
~LDialogBox()
FinishCreateSelf()
HandleKeyPress()
ListenToMessage()
SetCancelButton()
SetDefaultButton()

Data Members

The data members in this class are:

mDefaultButtonID
mCancelButtonID
mDefaultOutline

Operation

LDialogBox is a simple extension of the LWindow class, with the addition of button tracking for the default and cancel buttons. The data members that identify the default and cancel buttons are mDefaultButtonID and mCancelButtonID. You can set the buttons in the Constructor resource editor, or you can set them at runtime. You specify the button by its Pane ID number.

To learn more about using this class, refer to *The PowerPlant Book*.

Source files

(Pane Classes)

`LDialogBox.h`

`LDialogBox.cp`

Ancestors

LAttachable

LCommander

LListener

LModelObject

LPane

LView

LWindow

See Also

LControl

LDefaultOutline

LDialogBox()

Purpose The constructors create objects from the passed-in parameters.

Access Public

Prototype

```
LDialogBox();
LDialogBox( SWindowInfo &inWindowInfo );
LDialogBox( ResIDT inWINDid,
     Uint32 inAttributes,
     LCommander *inSuper );
LDialogBox( LStream *inStream );
```

Parameters The parameters for this constructor are:

`SWindowInfo&`	`inWindowInfo`	The reference to the window information for the SuperView.
`ResIDT`	`inWINDid`	The accompanying 'WIND' resource ID for this dialog box.
`Uint32`	`inAttributes`	A value that specifies the attributes for the window.

`LCommander*`	`inSuper`	A pointer to a SuperCommander.
`LStream*`	`inStream`	A pointer to a stream object that contains the information to create the LDialogBox object.

~LDialogBox()

Purpose The destructor destroys the object.

Access Public

Prototype

```
virtual ~LDialogBox();
```

FinishCreateSelf()

Purpose Finish creation by linking the dialog box as a Listener to its default and cancel buttons. This is an override of FinishCreateSelf() in LPane.

HandleKeyPress()

Purpose This method handles keyboard equivalents for hitting the default and cancel buttons of the dialog box. The default button is Enter or Return. The cancel button is Escape or Command-Period. This is an override of HandleKeyPress() in LCommander.

ListenToMessage()

Purpose This method respond to messages from Broadcasters. The dialog box responds to the `cmd_Close` message by deleting itself. This method is an override of ListenToMessage() in LListener.

Remarks Negative message numbers are relayed to the ProcessCommand(). This allows subclasses of the dialog box to handle messages as commands. Or, if you don't want to subclass this dialog box, the Super-

Commander of the dialog box will receive the messages as commands.

SetCancelButton()

Purpose Using this method you specify the Pane ID of the cancel button of the dialog box. The cancel button must be derived from LControl. This object is then added as a Listener.

Access Virtual, Public

Prototype

```
virtual void SetCancelButton( PaneIDT inButtonID );
```

Parameters The parameter for this method is:

`PaneIDT`	`inButtonID`	The pane ID of the cancel button.

Return None

Remarks Certain keyboard equivalents (Escape and Command-Period) simulate a click inside the cancel button.

SetDefaultButton()

Purpose Using this method you specify the Pane ID of the default button of the dialog box. The default button must be derived from LControl.

Access Virtual, Public

Prototype

```
virtual void SetDefaultButton( PaneIDT inButtonID);
```

Parameters The parameter for this method is:

`PaneIDT`	`inButtonID`	The pane ID of the default button.

Return None

Remarks The default button has a default outline around it and certain keyboard equivalents (Return and Enter) simulate a click inside it.

mDefaultButtonID

Purpose This data member stores the pane ID of the default button.

Access Protected

Prototype

```
PaneIDT mDefaultButtonID;
```

mCancelButtonID

Purpose This data member stores the pane ID of the cancel button.

Access Protected

Prototype

```
PaneIDT mCancelButtonID;
```

mDefaultOutline

Purpose This data mber stores the pointer to a LDefaultOutline.

Access Protected

Prototype

```
LDefaultOutline *mDefaultOutline;
```

LDocument

Overview

LDocument is a PowerPlant class that is used for opening, saving, closing, reverting, and printing a document. LDocument is an abstract class which provides a framework for supporting standard document operations via AppleEvents.

Methods

The methods in this class are:

LDocument()	~LDocument()
AskSaveAs()	AttemptClose()
AttemptQuitSelf()	Close()
DoAEClose()	DoAESave()
DoPrint()	DoRevert()
DoSave()	FindByFileSpec()
FindCommandStatus()	FindNamedDocument()
GetAEProperty()	GetDescriptor()
GetDocumentList()	HandleAppleEvent()
IsModified()	MakeCurrent()
MakeSelfSpecifier()	ObeyCommand()
SendAEClose()	SendAESaveAs()
SetModified()	UsesFileSpec()

Data Members

The data members in this class are:

sDocumentList	mPrintRecordH
mIsModified	mIsSpecified

Operation

You probably won't need to override all of these methods in a normal application, as many of them are complete.

Methods related to saving, reverting, and printing a document are empty, and need to be overrided.

Source files (Commander Classes)

`LDocument.h`

`LDocument.cp`

Ancestors LAttachable

LCommander

LModelObject

LDocument()

Purpose The constructors create objects from the passed-in parameters.

Access Public

Prototype

```
LDocument();
LDocument( LCommander *inSuper );
```

Parameters The parameter for these constructors is:

`LCommander*`	`inSuper`	The pointer to the Supercommander.

~LDocument()

Purpose The destructor destroys the object.

Access Virtual, Public

Prototype

```
virtual ~LDocument();
```

AskSaveAs()

Purpose Ask the user to save a Document and give it a name.

Access Virtual, Public

Prototype

```
virtual Boolean AskSaveAs(
     FSSpec &outFSSpec,
     Boolean inRecordIt );
```

Parameters The parameters for this method are:

`FSSpec&`	`outFSSpec`	The reference to the file we are working with.
`Boolean`	`inRecordIt`	Unused.

Return Returns `false` if the user cancels the operation, else it returns `true`.

AttemptClose()

Purpose Attempt to close a document. The Document might not close if it is modified and the user cancels the operation when asked whether to save the changes.

Access Virtual, Public

Prototype

```
virtual void AttemptClose( Boolean inRecordIt );
```

Parameters The parameter for this method is:

`Boolean`	`inRecordIt`	Unused.

Return None

AttemptQuitSelf()

Purpose Try to close a Document when quitting the program.

Access Virtual, Public

Prototype

```
virtual Boolean AttemptQuitSelf(
     Int32 inSaveOption );
```

Parameters The parameter for this method is:

`Int32`	`inSaveOption`	The save options for the document.

Return Return `false` if the user cancels when asked whether to save changes to a modified Document.

Close()

Purpose Close a document.

Access Public

Prototype

```
virtual void Close();
```

Parameters None

Return None

DoAEClose()

Purpose Close a document in response to a close AppleEvent.

Access Virtual, Public

Prototype

```
virtual void DoAEClose(
     const AppleEvent &inCloseAE );
```

Parameters The parameter for this method is:

`const AppleEvent&`	`inCloseAE`	The reference to the AppleEvent received.

Return None

DoAESave()

Purpose Save a Document in a particular file. This is equivalent to a "Save As" operation.

Subclasses should override this function to save the file, and should set mIsSpecified to `true`.

Access Virtual, Public

Prototype

```
virtual void DoAESave(
     FSSpec& inFileSpec,
     OSType inFileType );
```

Parameters The parameters for this method are:

`FSSpec&`	`inFileSpec`	The reference to the file we are working with.
`OSType`	`inFileType`	The resource type for the file.

Return None

DoPrint()

Purpose Print a document.

Access Virtual, Public

Prototype

```
virtual void DoPrint();
```

Parameters None

Return None

DoRevert()

Purpose Revert a document to its last saved version.

Access Virtual, Public

Prototype
```
virtual void DoRevert();
```

Parameters None

Return None

DoSave()

Purpose Save a Document, which must already be specified. Subclasses should override this method to save a document to an existing file.

Access Virtual, Public

Prototype
```
virtual void DoSave();
```

Parameters None

Return None

FindByFileSpec()

Purpose Return the document that uses the specified `FSSpec`.

Access Static, Public

Prototype
```
static LDocument* FindByFileSpec(
     const FSSpec &inFileSpec );
```

Parameters The parameters for this method are:

`const FSSpec&`	`inFileSpec`	The reference to the file we are working with.

Return A pointer to the LDocument that corresponds to the file we're interesting in.

FindCommandStatus()

Purpose Return whether a command is enabled and/or marked in a Menu. This method is an override of FindCommandStatus() in LCommander.

FindNamedDocument()

Purpose Return the document with the specified name.

Access Static, Public

Prototype

```
static LDocument* FindNamedDocument(
     ConstStringPtr inName );
```

Parameters The parameter for this method is:

`ConstStringPtr`	`inName`	The string pointer for the name to be found.

Return A pointer to the LDocument we are interested in, or `nil` if not found.

GetAEProperty()

Purpose This method returns an AppleEvent property.

Access Virtual, Public

Prototype

```
virtual void GetAEProperty(
     DescType inProperty,
     const AEDesc &inRequestedType,
     AEDesc &outPropertyDesc ) const;
```

Parameters The parameters for this method are:

`DescType`	`inProperty`	The descriptor type for the property.
`const AEDesc&`	`inRequestedType`	The AppleEvent descriptor type.
`AEDesc&`	`outPropertyDesc`	The property descriptor.

Return None

GetDescriptor()

Purpose This method is a pure virtual method that must be implemented in your subclass. It returns the AppleEvent descriptor.

Access Pure Virtual, Public

Prototype

```
virtual StringPtr GetDescriptor(
     Str255 outDescriptor ) const = 0;
```

Parameters The parameter for this method is:

`Str255`	`outDescriptor`	The string.

Return A `StringPtr` to the descriptor string.

GetDocumentList()

Purpose Returns the value of the sDocumentList data member.

Access Static, Public

Prototype

```
static TArray<LDocument*>& GetDocumentList();
```

Parameters None

Return Returns the value of the sDocumentList.

HandleAppleEvent()

Purpose This method handles an AppleEvent.

Access Virtual, Public

Prototype

```
virtual void HandleAppleEvent(
     const AppleEvent &inAppleEvent,
     AppleEvent &outAEReply,
     AEDesc &outResult,
     long inAENumber );
```

Parameters The parameters for this method are:

`const AppleEvent&`	`inAppleEvent`	The reference to the AppleEvent.
`AppleEvent&`	`outAEReply`	The AppleEvent reply.
`AEDesc&`	`outResult`	The AppleEvent result.
`long`	`inAENumber`	The AppleEvent number.

Return None

IsModified()

Purpose Return whether a document has been modified since it was last saved.

Access Virtual, Public

Prototype

```
virtual Boolean IsModified();
```

Parameters None

Return None

MakeCurrent()

Purpose Make this Document the current one. You should override this method to do what makes sense for a particular kind of document. In most cases, this means selecting the main window for a document.

Access Virtual, Public

Prototype
```
virtual void MakeCurrent();
```

Parameters None

Return None

MakeSelfSpecifier()

Purpose AppleEvent Object Model Support

Access Virtual, Public

Prototype
```
virtual void MakeSelfSpecifier(
    AEDesc &inSuperSpecifier,
    AEDesc &outSelfSpecifier ) const;
```

Parameters The parameters for this method are:

`AEDesc&`	`inSuperSpecifier`	The AppleEvent super specifier.
`AEDesc&`	`outSelfSpecifier`	The AppleEvent self specifier.

Return None

ObeyCommand()

Purpose Respond to commands.

Access Virtual, Public

Prototype

```
virtual Boolean ObeyCommand( CommandT inCommand,
    void *ioParam );
```

Parameters The parameters for this method are:

`CommandT`	`inCommand`	Command passed to this method.
`void*`	`ioParam`	The data block accompanying the command.

Return Return `true` if the command was handled, else return `false`.

SendAEClose()

Purpose Sends the close AppleEvent.

Access Virtual, Public

Prototype

```
virtual void SendAEClose(
    Int32 inSaveOption,
    FSSpec &inFileSpec,
    Boolean inExecute );
```

Parameters The parameters for this method are:

`Int32`	`inSaveOption`	The save options.
`FSSpec&`	`inFileSpec`	The reference to the file we are working with.
`Boolean`	`inExecute`	The value passed to `SendAppleEvent()`.

Return None

SendAESaveAs()

Purpose Sends the Save As AppleEvent.

Access Virtual, Public

Prototype

```
virtual void SendAESaveAs(
     FSSpec &inFileSpec,
     OSType inFileType,
     Boolean inExecute );
```

Parameters The parameters for this method are:

`OSType`	`inFileType`	The resource type for the file.
`FSSpec&`	`inFileSpec`	The reference to the file we are working with.
`Boolean`	`inExecute`	The value passed to `SendAppleEvent()`.

Return None

SetModified()

Purpose Specify whether a document has been modified since it was last saved.

Access Virtual, Public

Prototype

```
virtual void SetModified( Boolean inModified );
```

Parameters The parameter for this method is:

`Boolean`	`inModified`	The value to set in mIsModified.

Return None

UsesFileSpec()

Purpose Return whether the Document has a file that uses the specified `FSSpec`.

Access Virtual, Public

Prototype
```
virtual Boolean UsesFileSpec(
     const FSSpec& inFileSpec) const;
```

Parameters The parameter for this method is:

`const FSSpec&`	`inFileSpec`	The reference to the file we are working with.

Return Return `true` if the document has a file with the `FSSpec`, else return `false`.

sDocumentList

Purpose The static document list of pointers to LDocument objects.

Access Static, Protected

Prototype
```
static TArray<LDocument*> sDocumentList;
```

mPrintRecordH

Purpose The handle to the print record.

Access Protected

Prototype
```
THPrint mPrintRecordH;
```

mIsModified

Purpose Indicates whether a document is modified (dirty) or not.

Access Protected

Prototype
```
Boolean mIsModified;
```

mIsSpecified

Purpose Indicates whether a document has the specified information to save without interacting with the user first.

Access Protected

Prototype
```
Boolean mIsSpecified;
```

LDragAndDrop

Overview

LDragAndDrop is a PowerPlant mix-in class that is used for adding drag and drop features to a Pane.

Methods

The methods in this class are:

LDragAndDrop()
FocusDropArea()
HiliteDropArea()
PointInDropArea()

Data Members

The data members in this class are:

mPane

Operation

LDragAndDrop inherits from LDropArea and is a concrete implementation of that class. A pane that supports drag and drop should inherit from LDragAndDrop, not LDropArea.

For more information on drag and drop using PowerPlant, refer to *PowerPlant Advanced Topics*.

Source files

(Feature Classes)

`LDragAndDrop.h`

`LDragAndDrop.cp`

Ancestors

LDropArea

LDragAndDrop()

Purpose

This creates a drag and drop object from the passed-in parameters.

Access

Public

Prototype

```
LDragAndDrop( WindowPtr inMacWindow,
     LPane *inPane );
```

Parameters The parameters for this constructor are:

`WindowPtr`	`inMacWindow`	Toolbox window containing the drag and drop. It may be nil if the drag and drop is not in a window (e.g., if printing the drag and drop).
`LPane*`	`inPane`	Pane associated with the drag and drop. The drop area of the drag and drop is the Frame of the Pane.

FocusDropArea()

Purpose Set up a local coordinate system and clipping region for a drop area.

Access Virtual, Public

Prototype
```
virtual void FocusDropArea();
```

Parameters None

Return None

HiliteDropArea()

Purpose Hilite a drop area to indicate that it can accept the current drag.

For a drag and drop, the drop area is the Frame of its associated Pane inset by one pixel to account for the border which usually surrounds a drop-capable Pane.

Access Virtual, Protected

Prototype
```
virtual void HiliteDropArea(
     DragReference inDragRef);
```

Parameters The parameter for this method is:

`DragReference`	`inDragRef`	A reference to a drop area to hilite.

Return None

PointInDropArea()

Purpose Check whether a Point, in global coordinates, is inside a drop area.

Access Virtual, Public

Prototype
```
virtual Boolean PointInDropArea( Point inPoint );
```

Parameters The parameter for this method is:

`Point`	`inPoint`	A point to check.

Return `Boolean` `true` if the point is within the drop area, else `false`.

mPane

Purpose This data member stores the Pane that we're operating with.

Access Protected

Prototype
```
LPane *mPane;
```

LDragTask

Description LDragTask is a PowerPlant class that is used for encapsulating a single drag action initiated using the Mac OS Drag Manager.

Methods The methods in this class are:

LDragTask()	~LDragTask()
AddFlavors()	AddRectDragItem()
DoDrag()	DropLocationIsFinderTrash()
GetDragReference()	GetDragRegion()
InitDragTask()	MakeDragRegion()

Data Members The data members in this class are:

mDragRef	mDragRegion
mEventRecord	

Operation Normally, you will use this class in conjunction with the LDragAndDrop class. To learn more information about using this class, refer to *PowerPlant Advanced Topics*.

Source files (Feature Classes)

`LDragTask.h`

`LDragTask.cp`

See also LDragAndDrop

LDragTask()

Purpose The constructors create objects from the passed-in parameters.

Access Public

Prototypes

```
LDragTask( const EventRecord &inEventRecord );

// This constructor is for dragging a single, rectangular item
LDragTask( const EventRecord &inEventRecord,
     const Rect &inItemRect,
     ItemReference inItemRef,
     FlavorType inFlavor,
     void *inDataPtr,
     Size inDataSize,
     FlavorFlags inFlags );

// This constructor is for dragging an item outlined by a region
LDragTask( const EventRecord &inEventRecord,
     RgnHandle inItemRgn,
     ItemReference inItemRef,
     FlavorType inFlavor,
     void *inDataPtr,
     Size inDataSize,
     FlavorFlags inFlags );
```

Parameters These constructors have the following parameters:

`const EventRecord&`	`inEventRecord`	A reference to the Mac OS Toolbox mouse down `EventRecord`.
`const Rect&`	`inItemRect`	The Rect coordinates of the drag.
`ItemReference`	`inItemRef`	A reference to the item that is in the drag.
`FlavorType`	`inFlavor`	This is the "flavor," or data type for the drag.
`void*`	`inDataPtr`	This is a pointer to the data block for the drag.
`Size`	`inDataSize`	This is the size of the data block.
`FlavorFlags`	`inFlags`	This is a value for flavor flags, defined in the `Drag.h` Mac OS system header.
`RgnHandle`	`inItemRgn`	This must be in global coordinates, and will not be disposed of by this class.

Remarks The `inItemRect` is in the local coordinates of the current port, so you may need to call `SetPort` (or `FocusDraw`) beforehand.

~LDragTask()

Purpose The destructor destroys the object.

Access VIrtual, Public

Prototype

```
virtual ~LDragTask();
```

AddFlavors()

Purpose Add flavored items to the DragTask. If you use the short form of the LDragTask() constructor (`EventRecord` only), you must override this function to add items to the drag task.

Access Virtual, Protected

Prototype

```
void AddFlavors(DragReference inDragRef );
```

Parameters The parameter for this method is:

`DragReference`	`inDragRef`	A reference to an item to add to the drag.

Return None

AddRectDragItem()

Purpose This is a utility method for adding a rectangular item to the drag.

Access Virtual, Protected

Prototype

```
void AddRectDragItem( ItemReference inItemRef,
     const Rect &inItemRect);
```

Parameters This method has the following parameters:

`ItemReference`	`inItemRef`	A reference to the item to add to the drag.
`const Rect&`	`inItemRect`	The Rect must be in Global coordinates.

Return None

DoDrag()

Purpose This method performs a drag task. This function calls other member functions to add items to the drag and build the drag region, then calls the Drag Mananger to track the drag. You will not normally override this function.

Access Virtual, Public

Prototype
```
OSErr DoDrag();
```

Parameters None

Return This method returns an `OSErr` error code if an error occurs executing the drag. If no error occurs, the return value will be `noErr`.

DropLocationIsFinderTrash()

Purpose Return whether the drop location of the Drag is the Finder's Trash can Normally, you'll call this routine after the Drag completes to determine whether to delete the dragged items.

Access Public

Prototype
```
Boolean DropLocationIsFinderTrash();
```

Parameters None

Return A `Boolean` indicating whether the drop location of the drag is the Finder's trash can.

Remarks Normally, you'll call this routine after the drag completes to determine whether to delete the dragged items.

GetDragReference()

Purpose This method returns the value of mDragRef.

Access Public, Inline

Prototype
```
DragReference GetDragReference();
```

Parameters None

Return A `DragReference` containing the value of mDragRef.

GetDragRegion()

Purpose This method returns the value of mDragRegion.

Access Public, Inline

Prototype
```
RgnHandle GetDragRegion();
```

Parameters None

Return A `RgnHandle` containing the value of mDragRegion.

InitDragTask()

Purpose This method is a private initializer for the class.

Access Private

Prototype
```
void InitDragTask();
```

Parameters None

Return None

MakeDragRegion()

Purpose Build the region outlining the items to be dragged.

Access Virtual, Protected

Prototype

```
void MakeDragRegion( DragReference inDragRef,
     RgnHandle inDragRegion );
```

Parameters This method has the following parameters:

`DragReference`	`inDragRef`	A reference to the item in the drag.
`RgnHandle`	`inDragRegion`	The drag region for the item.

Return None

Remarks If you use the short form of the LDragTask() constructor (`EventRecord` only), you must override this function to specify the drag region for each item in the drag task.

mDragRef

Purpose This data member is the drag reference.

Access Protected

Prototype

```
DragReference mDragRef;
```

mDragRegion

Purpose This data member is the drag region.

Access Protected

Prototype
```
RgnHandle mDragRegion;
```

mEventRecord

Purpose This data member is the eent record, containing information about the event that just occurred.

Access Protected

Prototype
```
const EventRecord &mEventRecord;
```

LDropArea

Description

LDropArea is a PowerPlant abstract base class that is used for providing the interface to implement tracking and receiving functionality.

Methods

The methods in this class are:

LDropArea()
~LDropArea()
AddDropArea()
DoDragDrawing()
DoDragInput()
DoDragReceive()
DoDragSendData()
DragAndDropIsPresent()
DragIsAcceptable()
EnterDropArea()
FindDropArea()
FocusDropArea()
HandleDragDrawing()
HandleDragInput()
HandleDragReceive()
HandleDragSendData()
HandleDragTracking()
HiliteDropArea()
InTrackingWindow()
InsideDropArea()
InstallHandlers()
ItemIsAcceptable()
LeaveDropArea()
PointInDropArea()
ReceiveDragItem()
RemoveDropArea()
UnhiliteDropArea()

Data Members

The data members in this class are:

mDragWindow
mCanAcceptCurrentDrag
mIsHilited
sDragTrackingProc
sDragReceiveProc
sDropAreaList
sCurrentDropArea
sDragHasLeftSender

Operation LDropArea uses the LArray class. Most of the data members of LDropArea are used internally by PowerPlant, and you won't need to use them.

Source files (Feature Classes)

`LDragTask.h`

`LDragTask.cp`

See also LArray

LDragAndDrop

LDragTask

LDropArea()

Purpose The constructor creates an object from the passed-in parameters. It initializes drag and drop if needed.

Access Public

Prototype

```
LDropArea ( WindowPtr inWindow );
```

Parameters This constructor has the following parameter:

`WindowPtr`	`inWindow`	The window in which to establish a drop area.

~LDropArea()

Purpose The destructor destroys the object.

Access Virtual, Public

Prototype

```
virtual ~LDropArea();
```

AddDropArea()

Purpose Add a drop area to the list maintained by this class.

Access Protected, Static

Prototype

```
void AddDropArea( LDropArea *inDropArea,
     WindowPtr inMacWindow );
```

Parameters This method has the following parameter:

`LDropArea*`	`inDropArea`	A pointer to the drop area object to add to the list.
`WindowPtr`	`inMacWindow`	The window pointer for the drop area to be added.

Return None

DoDragDrawing()

Purpose Draw the items for a drag in progress. This method gets called if you installed the optional DragDrawingProc for this drop area.

Access Virtual, Protected

Prototype

```
virtual void DoDragDrawing(
     DragRegionMessage inMessage,
     RgnHandle inShowRgn,
     Point inShowOrigin,
     RgnHandle inHideRgn,
     Point inHideOrigin,
     DragReference inDragRef );
```

Parameters This method has the following parameter:

`DragRegion-Message`	`inMessage`	The message passed to this object from the framework.
`RgnHandle`	`inShowRgn`	The show region.
`Point`	`inShowOrigin`	The origin point.
`RgnHandle`	`inHideRgn`	The hide region.
`Point`	`inHideOrigin`	The hide origin point.
`DragReference`	`inDragRef`	The reference to the area.

Return None

DoDragInput()

Purpose Modify the state of the mouse and modifier keys during a drag. This method gets called if you installed the optional DragInputProc for this drop area.

Access Virtual, Protected

Prototype

```
virtual void DoDragInput( Point ioMouse,
     Int16 ioModifiers,
     DragReference inDragRef );
```

Parameters This method has the following parameter:

`Point`	`ioMouse`	The point to check against.
`Int16`	`ioModifiers`	The modifier keys, if any.
`DragReference`	`inDragRef`	The reference to the area.

Return None

DoDragReceive()

Purpose Receive the items from a completed drag and drop. This method gets called after items are dropped into a drop area. The drag contains items that are acceptable, as defined by the and member functions. This method repeatedly calls ReceiveDragItem() for each item in the drag. You should override this method if you want to process the dragged items all at once.

Access Virtual, Protected

Prototype

```
virtual void DoDragReceive(
     DragReference inDragRef );
```

Parameters This method has the following parameter:

`DragReference`	`inDragRef`	The reference to the area.

Return None

DoDragSendData()

Purpose Send the data associated with a particular drag item. This method gets called if you installed the optional DragSendDataProc for this drop area, in which case you should override this function to provide the requested data by calling `SetDragItemFlavorData()`.

Access Virtual, Protected

Prototype

```
virtual void DoDragSendData( FlavorType inFlavor,
     ItemReference inItemRef,
     DragReference inDragRef );
```

Parameters This method has the following parameters:

`FlavorType`	`inFlavor`	This is the "flavor," or data type for the drag.
`ItemReference`	`inItemRef`	A reference to the item that is in the drag.
`DragReference`	`inDragRef`	The reference to the area.

Return None

DragAndDropIsPresent()

Purpose Indicates whether drag and drop is implemented on the running Mac OS system.

Access Static, Public

Prototype
```
static Boolean DragAndDropIsPresent();
```

Parameters None

Return `Boolean` of `true` if drag and drop is installed, else `false`.

DragIsAcceptable()

Purpose Indicate whether a drop area can accept the specified drag. A drag is acceptable if all items in the drag are acceptable.

Access Virtual, Protected

Prototype
```
virtual Boolean DragIsAcceptable(
    DragReference inDragRef);
```

Parameters This method has the following parameter:

`DragReference`	`inDragRef`	The reference to the area.

Return `Boolean` of `true` if the drop area can accept the drag, else `false`.

EnterDropArea()

Purpose This method is called when a drag is entering a drop area. This call will be followed by a corresponding LeaveDropArea() call (when the drag moves out of the drop area or after the drag is received by this drop area). If the drop area can accept the drag and the drag is coming from outside the drop area, hilite the drop area.

Access Virtual, Protected

Prototype

```
virtual void EnterDropArea( DragReference
     inDragRef,
     Boolean inDragHasLeftSender);
```

Parameters This method has the following parameters:

`DragReference`	`inDragRef`	The reference to the area.
`Boolean`	`inDragHasLeft-Sender`	

Return None

FindDropArea()

Purpose Return the drop area in the specified window containing the specified point which can receive the drag.

Access Static, Protected

Prototype

```
static LDropArea* FindDropArea(
     WindowPtr inMacWindow,
     Point inGlobalPt,
     DragReference inDragRef);
```

Parameters This method has the following parameters:

`WindowPtr`	`inMacWindow`	The window to search in.
`Point`	`inGlobalPt`	The point (in global coordinates).
`DragReference`	`inDragRef`	The reference to the area.

Return Returns `nil` if no drop area meets the requirements, else returns a pointer to the drop area.

FocusDropArea()

Purpose Set up local coordinate system and clipping region for a drop area. The Drag Manager sets the port to the Window containing the drop rectangle. However, it doesn't know anything about the local coordinates and clipping region.

Access Virtual, Public

Prototype
```
void FocusDropArea();
```

Parameters None

Return None

HandleDragDrawing()

Purpose Drag Manager callback for drawing the items during a drag.

Access Static, Protected

Prototype
```
static pascal OSErr HandleDragDrawing(
     DragRegionMessage inMessage,
     RgnHandle inShowRgn,
     Point inShowOrigin,
     RgnHandle inHideRgn,
     Point inHideOrigin,
```

```
    void *inRefCon,
    DragReference inDragRef);
```

Parameters This method has the following parameters:

`DragRegion-Message`	`inMessage`	The message passed to this method from the framework.
`RgnHandle`	`inShowRgn`	The show region.
`Point`	`inShowOrigin`	The point in the show region.
`RgnHandle`	`inHideRgn`	The hide region.
`Point`	`inHideOrigin`	The point in the hide region.
`void*`	`inRefCon`	A pointer to the Mac OS refcon.
`DragReference`	`inDragRef`	The reference to the area.

Return Returns an `OSErr` value, which will be `noErr` if everything succeeds.

HandleDragInput()

Purpose This is a Drag Manager callback for manipulating the mouse and modifier keys during a drag. This is basically an exception-handling wrapper for calling DoDragInput().

Access Static, Protected

Prototype

```
static pascal OSErr HandleDragInput(
    Point *ioMouse,
    Int16 *ioModifiers,
    void *inRefCon,
    DragReference inDragRef);
```

Parameters This method has the following parameters:

`Point*`	`ioMouse`	The pointer to a point structure.
`Int16*`	`ioModifiers`	The show region.
`void*`	`inRefCon`	The point in the show region.
`DragReference`	`inDragRef`	The reference to the area.

Return Returns an `OSErr` value, which will be `noErr` if everything succeeds.

HandleDragReceive()

Purpose This is a Drag Manager callback for receiving a successful drop.

Access Static, Protected

Prototype

```
static pascal OSErr HandleDragReceive(
    WindowPtr inMacWindow,
    void* inRefCon,
    DragReference inDragRef);
```

Parameters This method has the following parameters:

`WindowPtr`	`inMacWindow`	The pointer to the window receiving the drag.
`void*`	`inRefCon`	The Mac OS refcon pointer.
`DragReference`	`inDragRef`	The reference to the area.

Return Returns an `OSErr` value, which will be `noErr` if everything succeeds.

HandleDragSendData()

Purpose This is a Drag Manager callback for sending the data for an item that is part of an accepted drag and drop.

Access Static, Protected

Prototype

```
static pascal OSErr HandleDragSendData(
     FlavorType inFlavor,
     void *inRefCon,
     ItemReference inItemRef,
     DragReference inDragRef);
```

Parameters This method has the following parameters:

`FlavorType`	`inFlavor`	The flavor type of the drag and drop (enumerated in `Drag.h`).
`void*`	`inRefCon`	The Mac OS refcon pointer.
`ItemReference`	`inItemRef`	The reference to the item that was drag and dropped.
`DragReference`	`inDragRef`	The reference to the area.

Return Returns an `OSErr` value, which will be `noErr` if everything succeeds.

HandleDragTracking()

Purpose This is the Drag Manager callback for tracking a drag in progress.

Access Static, Protected

Prototype

```
static pascal OSErr HandleDragTracking(
     DragTrackingMessage inMessage,
     WindowPtr inMacWindow,
     void* inRefCon,
     DragReference inDragRef);
```

Parameters This method has the following parameters:

`DragTracking-Message`	`inMessage`	The message that is passed to this method.
`WindowPtr`	`inMacWindow`	The window pointer.
`void*`	`inRefCon`	The Mac OS refcon pointer.
`DragReference`	`inDragRef`	The reference to the area.

Return Returns an `OSErr` value, which will be `noErr` if everything succeeds.

HiliteDropArea()

Purpose Hilite a drop area to indicate that it can accept the current drag. Any subclasses should override this function to create a region representing the drop area, and call `ShowDragHilite()`.

Access Virtual, Protected

Prototype

```
virtual void HiliteDropArea(
     DragReference inDragRef );
```

Parameters This method has the following parameter:

`DragReference`	`inDragRef`	The reference to the area.

Return None

InTrackingWindow()

Purpose Track a Drag while it is inside a Window.

Access Static, Protected

Prototype

```
static void InTrackingWindow(
     WindowPtr inMacWindow,
     DragReference inDragRef);
```

Parameters This method has the following parameters:

`WindowPtr`	`inMacWindow`	The window pointer.
`DragReference`	`inDragRef`	The reference to the area.

Return None

InsideDropArea()

Purpose Track a Drag while it is inside a drop area. This method is called repeatedly while an acceptable drag is inside a drop area. Any subclasses may override to provide additional visual feedback during a drag, such as indicating an insertion point

Access Virtual, Protected

Prototype

```
virtual void InsideDropArea(
     DragReference inDragRef);
```

Parameters This method has the following parameter:

`DragReference`	`inDragRef`	The reference to the area.

Return None

InstallHandlers()

Purpose Install Tracking and Receive Handlers for the Drag Manager. We use a single Tracking handler and a single Receive handler for all drag and drop operations. With the Drag Manager, you can register handlers globally or for specific windows. With PowerPlant, we want to have separate handlers for each Pane that supports drag

and drop. Therefore, this class maintains a list of all drag and drop Panes and dispatches calls to the proper Panes.

Access Static, Protected

Prototype
```
static void InstallHandlers();
```

Parameters None

Return None

ItemIsAcceptable()

Purpose This method checks for each acceptable flavor, exiting immediately-upon finding an acceptable flavor.

Access Virtual, Protected

Prototype
```
virtual Boolean ItemIsAcceptable(
     DragReference inDragRef,
     ItemReference inItemRef );
```

Parameters This method has the following parameters:

`DragReference`	`inDragRef`	The reference to the area.
`ItemReference`	`inItemRef`	The reference to the drag item.

Return `Boolean` returning `true` if the item is acceptable, `false` if it is not.

LeaveDropArea()

Purpose This method is called when a drag is leaving a drop area. This call will have been preceded by a corresponding EnterDropArea() call.

Access Virtual, Protected

Prototype
```
virtual void LeaveDropArea(DragReference
     inDragRef);
```

Parameters This method has the following parameter:

`DragReference`	`inDragRef`	The reference to the area.

Return None

Remarks Remove hiliting of the drop area if necessary.

PointInDropArea()

Purpose This method takes a point and indicates whether the point lies within the drop area or not.

Access Pure Virtual, Public

Prototype

```
virtual Boolean PointInDropArea(
     Point inGlobalPt) = 0;
```

Parameters This method has the following parameter:

`Point`	`inGlobalPt`	The point to check.

Return `Boolean` of `true` if the point is in the drop area, else `false`.

Remarks This method is a pure virtual method, and your subclass must implement it.

ReceiveDragItem()

Purpose Process an item which has been dragged into a drop area. This method gets called once for each item contained in a completed drag. The item will have returned true from ItemIsAcceptable(). The drop area is focused upon entry and `inItemBounds` is specified in the local coordinates of the drop area. Override this function if the drop area can accept dropped items. You may want to call `GetFlavorData()` and `GetFlavorDataSize()` if there is information associated with the dragged Item.

Access Virtual, Protected

Prototype

```
virtual, void ReceiveDragItem(
     DragReference inDragRef,
     DragAttributes inDragAttrs,
     ItemReference inItemRef,
     Rect& inItemBounds );
```

Parameters This method has the following parameters:

`DragReference`	`inDragRef`	The reference to the drag.
`DragAttributes`	`inDragAttrs`	The drag attributes.
`ItemReference`	`inItemRef`	The reference to the item that was drag and dropped.
`Rect&`	`inItemBounds`	The Rect in local coordinates.

Return None

RemoveDropArea()

Purpose Remove a drop area from the list maintained by this class.

Access Static, Protected

Prototype

```
static void RemoveDropArea(
     LDropArea *inDropArea,
     WindowPtr inMacWindow);
```

Parameters This method has the following parameters:

`LDropArea*`	`inDropArea`	The window pointer.
`WindowPtr`	`inMacWindow`	The reference to the area.

Return None

UnhiliteDropArea()

Purpose Unhilite a drop area when a drag leaves the area. Subclasses should override this function if they override HiliteDropArea() to do something other than call `ShowDragHilite()`.

Access Virtual, Protected

Prototype

```
virtual void UnhiliteDropArea(
     DragReference inDragRef);
```

Parameters This method has the following parameter:

`DragReference`	`inDragRef`	The reference to the area.

Return None

mDragWindow

Purpose This data member is the `WindowPtr` for the window containing the drop area.

Access Protected

Prototype

```
WindowPtr mDragWindow;
```

mCanAcceptCurrentDrag

Purpose This data member indicates whether the drop area can receive the drag.

Access Protected

Prototype

```
Boolean mCanAcceptCurrentDrag;
```

mIsHilited

Purpose This data member indicates whether the drop area is currently highlighted.

Access Protected

Prototype
```
Boolean mIsHilited;
```

sDragTrackingProc

Purpose This data member stores the procedure pointer for the drag tracking procedure.

Access Static

Prototype
```
static DragTrackingHandlerUPP sDragTrackingProc;
```

sDragReceiveProc

Purpose This data member stores the procedure pointer for the drag receive procedure.

Access Static

Prototype
```
static DragReceiveHandlerUPP sDragReceiveProc;
```

sDropAreaList

Purpose This data member stores the drop area list.

Access Static

Prototype
```
static TArray<SDropAreaEntry> *sDropAreaList;
```

sCurrentDropArea

Purpose This data member stores the current drop area.

Access Static

Prototype

```
static LDropArea *sCurrentDropArea;
```

sDragHasLeftSender

Purpose This data member indicates whether the drag has left the sender window.

Access Static

Prototype

```
static Boolean sDragHasLeftSender;
```

LdtFApplication

Overview LdtFApplication is a PowerPlant class that

Methods The methods in this class are:

LdtFApplication() ~LdtFApplication()

OpenDatabase() GetFileWithPrompt()

Source files (_Advanced Classes:Database Access)

`LdtFApplication.h`

`LdtFApplication.cp`

Ancestors LDBApplication

See Also LODBCApplication

LdtFApplication()

Purpose The default constructor initializes the object.

Access Public

Prototype
```
LdtFApplication();
```

Parameters None

~LdtFApplication()

Purpose The destructor destroys the object.

Access Public, Virtual

Prototype
```
virtual ~LdtFApplication();
```

Parameters None

OpenDatabase()

Purpose Opens the database.

Access Virtual, Public

Prototype

```
virtual void OpenDatabase();
```

Parameters None

Return None

GetFileWithPrompt()

Purpose Call the standard file open dialog.

Access Virtual, Public

Prototype

```
virtual voidGetFileWithPrompt(
    Str255  inPrompt,
    ResType inType,
    StandardFileReply&outReply);
```

Parameters The parameters for this member function are:

`inPrompt`	The string to display in the dialog box.
`inType`	The resource type.
`outReply`	The standard file dialog data structure.

Return None

LEditField

Overview

LEditField is a PowerPlant class that encapsulates the behavior of an editable text field, as commonly appears in a dialog box.

Methods

The methods in this class are:

LEditField()
~LEditField()
AdjustCursorSelf()
AdjustTextWidth()
AlignTextEditRects()
BeTarget()
ClickSelf()
DisableSelf()
DontBeTarget()
DrawBox()
DrawSelf()
EnableSelf()
FindCommandStatus()
FocusDraw()
GetDescriptor()
GetMacTEH()
GetValue()
HandleKeyPress()
HideSelf()
InitEditField()
MoveBy()
ObeyCommand()
ResizeFrameBy()
RestorePlace()
SavePlace()
SelectAll()
SetDescriptor()
SetKeyFilter()
SetMaxChars()
SetTextTraitsID()
SetValue()
SpendTime()
TooManyCharacters()
UseWordWrap()
UserChangedText()

Data Members The data members in this class are:

mTextEditH	mKeyFilter
mTypingAction	mMaxChars
mTextTraitsID	mHasBox
mHasWordWrap	

Operation LEditField uses TextEdit to implement an editable text field. PowerPlant also handles undo and redo for most text-related actions.

Source files (Pane Classes)

`LEditField.h`

`LEditField.cp`

Ancestors LAttachable

LCommander

LPane

LPeriodical

LEditField()

Purpose The constructors create objects from the passed-in parameters.

Access Public

Prototype

```
LEditField();
LEditField( const LEditField &inOriginal );
LEditField( const SPaneInfo &inPaneInfo,
     Str255 inString,
     ResIDT inTextTraitsID,
     Int16 inMaxChars,
     Boolean inHasBox,
     Boolean inHasWordWrap,
```

```
    KeyFilterFunc inKeyFilter,
    LCommander *inSuper );
LEditField( const SPaneInfo &inPaneInfo,
    Str255 inString,
    ResIDT inTextTraitsID,
    Int16 inMaxChars,
    Uint8 inAttributes,
    KeyFilterFunc inKeyFilter,
    LCommander *inSuper );
LEditField( LStream *inStream );
```

Parameters The parameters for these constructors are:

`const LEditField&`	`inOriginal`	The reference to the object to copy.
`const SPaneInfo&`	`inPaneInfo`	Reference to the Superpane info.
`Str255`	`inString`	The string for the field.
`ResIDT`	`inTextTraitsID`	The text traits resource ID.
`Int16`	`inMaxChars`	The max number of characters.
`Boolean`	`inHasBox`	Indicates whether to see the `editAttr_Box` attribute or not.
`Boolean`	`inHasWordWrap`	Indicates whether the text field wraps. This sets the `editAttr_WordWrap` attribute.
`KeyFilterFunc`	`inKeyFilter`	A pointer to a key press filter procedure.
`LCommander*`	`inSuper`	The Supercommander.
`Uint8`	`inAttributes`	The attributes

~LEditField()

Purpose The destructor destroys the object.

Access Virtual, Public

Prototype

```
virtual ~LEditField();
```

AdjustCursorSelf()

Purpose EditField uses the standard I-Beam cursor. This is an override of AdjustCursorSelf() in LPane.

AdjustTextWidth()

Purpose Adjust the width of the destination rectangle of the Toolbox TextEdit record.

This function does nothing if mHasWordWrap is `true`. If mHasWordWrap is `false`, this function sets the width of the TextEdit destination rectangle to either the width of the text or a very large number, depending on the value of `inShrinkToText`.

This adjustment is needed to make autoscrolling work properly when mHasWordWrap is off. While entering text, the destination rectangle should be very wide so that the text doesn't word wrap. However, while clicking, it should be just as wide as the text so that the EditField does not autoscroll past the edge of the text.

Access Virtual, Protected

Prototype

```
virtual void AdjustTextWidth(
      Boolean inShrinkToText );
```

Parameters The parameter for this method is:

`Boolean`	`inShrinkToText`	Indicates whether to adjust the rectangle or not.

Return None

AlignTextEditRects()

Purpose Align the view and destination rectangles of the Toolbox TextEdit record with the Frame of an EditField.

Access Virtual, Protected

Prototype `virtual void AlignTextEditRects();`

Parameters None

Return None

BeTarget()

Purpose EditField is becoming the Target. This is an override of BeTarget() in LCommander.

ClickSelf()

Purpose Respond to Click inside an EditField. This is an override of ClickSelf() in LPane.

DisableSelf()

Purpose Disable an EditField. This is an override of DisableSelf() in LPane.

DontBeTarget()

Purpose EditField is becoming the Target. This is an override of DontBeTarget() in LCommander.

DrawBox()

Purpose Draw a box around an EditField.

Access Virtual, Protected

Prototype `virtual void DrawBox();`

Parameters None

Return None

Remarks Box around an EditField is outset from the Text by 2 pixels. The box itself is 1 pixel thick, drawn in the foreground color of the Pane (not necessarily the same as the text color). If the EditField is disabled, the box draws with a gray pattern. The 1 pixel rectangle between the box and the text draws in the background color of the text.

DrawSelf()

Purpose Draw an EditField. This is an override of DrawSelf() in LPane.

EnableSelf()

Purpose Enable an EditField. This is an override of EnableSelf() in LPane.

FindCommandStatus()

Purpose Pass back the status of a Command. This is an override of FindCommandStatus() in LCommander.

FocusDraw()

Purpose Prepare for drawing in the EditField. This is an override of FocusDraw() in LPane.

GetDescriptor()

Purpose Return the first 255 characters of the EditField as a Pascal string. The caller must allocate a Str255 variable for storing the string. This is an override of GetDescriptor() in LPane.

GetMacTEH()

Purpose This method returns the handle to the Mac OS TextEdit structure.

Access Public

Prototype

```
TEHandle GetMacTEH();
```

Parameters None

Return A handle to the TextEdit structure.

GetValue()

Purpose Return the integer value represented by the contents of an EditField. An empty or non-numerical EditField evaluates to zero. This is an override of GetValue() in LPane.

HandleKeyPress()

Purpose Handle key stroke directed at an EditField. Return true if the EditField handles the keystroke. This is an override of HandleKeyPress() in LCommander.

HideSelf()

Purpose Hide an EditField. An invisible EditField can't be on duty. This is an override of HideSelf() in LPane.

InitEditField()

Purpose Initialize member variables of a EditField to default values.

Access Private

Prototype

```
void InitEditField( Uint8 inAttributes );
```

Parameters The parameter for this method is:

`Uint8`	`inAttributes`	The attributes, from the enum values in `LEditField.h`

Return None

MoveBy()

Purpose Move the location of the Frame by the specified amounts. This is an override of MoveBy() in LPane.

ObeyCommand()

Purpose Handle standard editing commands. This is an override of ObeyCommand() in LCommander.

ResizeFrameBy()

Purpose Change the Frame size by the specified amounts `inWidthDelta` and `inHeightDelta` specify, in pixels, how much larger to make the Frame. Positive deltas increase the size, negative deltas reduce the size. This is an override of ResizeFrameBy() in LPane.

RestorePlace()

Purpose Save TextEdit rectangles. This is an override of RestorePlace() in LPane.

SavePlace()

Purpose Save TextEdit rectangles. This is an override of SavePlace() in LPane.

SelectAll()

Purpose Select entire contents of an EditField.

Access Public

Prototype
```
virtual void SelectAll();
```

Parameters None

Return None

SetDescriptor()

Purpose Set the contents of an EditField from a Pascal string. This is an override of SetDescriptor() in LPane.

SetKeyFilter()

Purpose Specify the function for filtering keystrokes.

Access Virtual, Public

Prototype

```
virtual void SetKeyFilter(
     KeyFilterFunc inKeyFilter );
```

Parameters The parameter for this method is:

`KeyFilterFunc`	`inKeyFilter`	The function pointer for the key filter proc.

Return None

SetMaxChars()

Purpose Specify the maximum number of characters that an EditField can contain.

Access Virtual, Public

Prototype

```
virtual void SetMaxChars( Int16 inMaxChars );
```

Parameters The parameter for this method is:

`Int16`	`inMaxChars`	The maximum number of characters.

Return None

SetTextTraitsID()

Purpose Specify the resource ID of the TextTraits for an EditField. This function updates the line height to fit the text characteristics.

Access Virtual, Public

Prototype

```
virtual void SetTextTraitsID(ResIDT
     inTextTraitsID);
```

Parameters The parameter for this method is:

`ResIDT`	`inTextTraitsID`	The resource ID for the text traits resource.

Return None

SetValue()

Purpose Set the contents of an EditField to the string representation of aspecified integer number. This is an override of SetValue() in LPane.

SpendTime()

Purpose At idle time, flash the insertion cursor. This is an override of SpendTime() in LPeriodical.

TooManyCharacters()

Purpose Indicate whether adding the specified number of characters will exceed the maximum allowed. This method assumes that the characters being added will replace the current selection.

Access Virtual, Protected

Prototype

```
virtual Boolean TooManyCharacters(
     Int32 inCharsToAdd );
```

Parameters The parameter for this method is:

`Int32`	`inCharsToAdd`	The number of characters you want to test for the maximal limit.

Return Returns `true` if the maximum will be exceeded, else `false`.

UseWordWrap()

Purpose Specify whether the EditField word wraps to its frame.

Access Virtual, Public

Prototype

```
virtual void UseWordWrap( Boolean inSetting );
```

Parameters The parameter for this method is:

`Boolean`	`inSetting`	Indicate whether to wrap or not.

Return None

UserChangedText()

Purpose Text of EditField has changed as a result of the user actions. You should override this to validate the field and/or dynamically update the field as the user types. This function is not called by SetDescriptor(), which is typically used to programatically change the text.

Access Virtual, Public

Prototype

```
virtual void UserChangedText();
```

Parameters None

Return None

mTextEditH

Purpose The handle to the TextEdit record.

Access Protected

Prototype
```
TEHandle mTextEditH;
```

mKeyFilter

Purpose The key filter proc storage space.

Access Protected

Prototype
```
KeyFilterFunc mKeyFilter;
```

mTypingAction

Purpose Pointer to an LTETypingAction object.

Access Protected

Prototype
```
LTETypingAction *mTypingAction;
```

mMaxChars

Purpose The maximum number of characters that the field can accomodate.

Access Protected

Prototype
```
Int16 mMaxChars;
```

mTextTraitsID

Purpose The resource ID for the text traits.

Access Protected

Prototype `ResIDT mTextTraitsID;`

mHasBox

Purpose Indicates whether the `editAttr_Box` enum attribute is set.

Access Protected

Prototype `Boolean mHasBox;`

mHasWordWrap

Purpose Indicates whether the `editAttr_WordWrap` enum attribute is set.

Access Protected

Prototype `Boolean mHasWordWrap;`

LEndpoint

Overview

LEndpoint is a PowerPlant class that forms the basis for networking. Most of the methods are pure virtual, which means that you need to supply implementations if you inherit from this class.

Methods

The methods in this class are:

LEndpoint()
~LEndpoint()
AbortThreadOperation()
AckSends()
Bind()
DontAckSends()
DontQueueSends()
GetLocalAddress()
GetState()
IsAckingSends()
IsQueuingSends()
QueueSends()
Unbind()

Data Members

The data members in this class are:

mQueueSends

Operation

This class encapsulates the idea of a "network endpoint," or one side of a two-way communication link. The endpoint is the object that is used to send or receive data over the network.

LEndpoint is an abstract base class (thus its constructor is declared protected). Use one of the predefined subclasses, such as LMacTCPTCPEndpoint or LOpenTptTCPEndpoint, or use the UNetworkFactory::CreateTCPEndpoint() function to create the appropriate endpoint for the system software that's installed on the user's machine.

Source files

(Networking Classes)

`LEndpoint.h`

`LEndpoint.cp`

See also

LBroadcaster

LMacTCPTCPEndpoint

LOpenTptTCPEndpoint

UNetworkFactory

LEndpoint()

Purpose The constructor creates the object.

Access Public

Prototype
```
LEndpoint();
```

Parameters None

~LEndpoint()

Purpose The destructor destroys the object.

Access Virtual, Public

Prototype
```
virtual ~LEndpoint();
```

Parameters None

AbortThreadOperation()

Purpose This method must be implemented by your class since it is pure virtual. Abort a thread operation.

Access Pure virtual, Public

Prototype
```
virtual voidAbortThreadOperation(
     LThread * inThread ) = 0;
```

Parameters The parameter for this methods is:

`LThread*`	`inThread`	`The pointer to the thread.`

Return None

AckSends()

Purpose Enable the acknowledgement of sent data for the endpoint.

Access Pure virtual, Public

Prototype

```
virtual voidAckSends() = 0;
```

Parameters None

Return None

Bind()

Purpose Reserve a TCP port for the connection.

Access Pure virtual, Public

Prototype

```
virtual voidBind(
     LInternetAddress& inLocalAddress,
     Uint32 inListenQueueSize = 0,
     Boolean inReusePort = true) = 0;
```

Parameters The parameters for this method are:

LInternetAddress&	inLocalAddress	The local address.
Uint32	inListenQueueSize	The listen queue size in bytes, the default is 0.
Boolean	inReusePort	Whether to reuse the port or not, the default is 0.

Return None

DontAckSends()

Purpose Disable the acknowledgement of sent data for the endpoint.

Access Pure virtual, Public

Prototype
```
virtual voidDontAckSends() = 0;
```

Parameters None

Return None

DontQueueSends()

Purpose This method sets the value of the mQueueSends data member to false.

Access Inline, Public, Virtual

Prototype
```
virtual void DontQueueSends();
```

Parameters None

Return None

GetLocalAddress()

Purpose Return the network address of your local computer.

Access Pure virtual, Public

Prototype
```
virtual LInternetAddress*GetLocalAddress() = 0;
```

Parameters None

Return A pointer to the address.

GetState()

Purpose Return the current state of the endpoint.

Access Pure virtual, Public

Prototype
```
virtual EEndpointStateGetState() = 0;
```

Parameters None

Return None

IsAckingSends()

Purpose Dtermine whether the endpoint is acknowledging sends.

Access Pure virtual, Public

Prototype
```
virtual BooleanIsAckingSends() = 0;
```

Parameters None

Return Returns true if the endpoint is ACK-ing sends.

IsQueuingSends()

Purpose Determines if the endpoint is queuing sends.

Access Virtual, Public

Prototype
```
virtual BooleanIsQueuingSends();
```

Parameters None

Return Returns true if the endpoint is queuing sends.

QueueSends()

Purpose Directs the endpoint to queue sends.

Access Virtual, Public

Prototype
```
virtual voidQueueSends();
```

Parameters None

Return None

Unbind()

Purpose Release a previously-bound TCP port.

Access Pure virtual, Public

Prototype
```
virtual voidUnbind() = 0;
```

Parameters None

Return None

mQueueSends

Purpose State variable for whether to queue send traffic for the endpoint.

Access Protected

Prototype
```
Boolean mQueueSends;
```

LEraseAttachment

Overview LEraseAttachment is a PowerPlant class that is used for erasing the Frame of a Pane.

Methods The methods in this class are:

LEraseAttachment() ExecuteSelf()

Data Members There are no data members for this class.

Operation This Attachment is designed to respond to the `msg_DrawOrPrint` message. When you create this Attachment, you specify whether the host should also draw.

Source files (Utility Classes)

`UAttachments.h`

`UAttachments.cp`

Ancestors LAttachment

LPane

See Also LPaintAttachment

LEraseAttachment()

Purpose The constructor creates the object.

Access Public

Prototype

```
LEraseAttachment( Boolean inExecuteHost );
LEraseAttachment( LStream *inStream );
```

Parameters The parameters for these constructors are:

Boolean	`inExecuteHost`	A Boolean indicating the value to be set for mExecuteHost.
`LStream*`	`inStream`	A pointer to a stream object that contains the information to create the LEraseAttachment object.

ExecuteSelf()

Purpose Erases the Frame of a Pane. This is an override of ExecuteSelf() in LAttachment.

LEventDispatcher

Overview

LEventDispatcher is a PowerPlant class that is used for handling all event processing after the application retrieves the event in the main event loop. This class is also responsible for adjusting the cursor, distributing time to idle-time processes, and initiating menu updates before displaying menus.

Methods

The methods in this class are:

LEventDispatcher()	~LEventDispatcher()
AdjustCursor()	ClickMenuBar()
DispatchEvent()	EventActivate()
EventAutoKey()	EventDisk()
EventHighLevel()	EventKeyDown()
EventKeyUp()	EventMouseDown()
EventMouseUp()	EventOS()
EventResume()	EventSuspend()
EventUpdate()	ExecuteAttachments()
GetCurrentEventDispatcher()	UpdateMenus()
UseIdleTime()	

Data Members

The data members in this class are:

sCurrentDispatcher	mSaveDispatcher
mMouseRgnH	

Operation

Most of the time, you won't need to override any functions in this class. PowerPlant supplies default behaviors that implement much of the functionality you could hope for.

Unless you use the Mac Toolbox directly for tasks such as running a dialog box with `ModelDialog()`, all event processing goes through the main event loop and LEventDIspatcher, even for modal dialogs.

Source files (Features Classes)

`LEventDispatcher.h`

`LEventDispatcher.cp`

See also LAttachable

LEventDispatcher()

Purpose The constructor creates the object, initializing the menus so that they are unhilited, and saving a copy of the current Event Dispatcher.

Access Public

Prototype

```
LEventDispatcher();
```

Parameters None

~LEventDispatcher()

Purpose The destructor destroys the object, including disposal of the mouse region, and restoring of the previous Event Dispatcher.

Access Virtual, Public

Prototype

```
virtual ~LEventDispatcher();
```

Parameters None

AdjustCursor()

Purpose Adjust the shape of the cursor (mouse pointer). If the cursor is inside an active, enabled, PowerPlant window, then the window is expected to handle cursor adjustment, using

Access Virtual, Protected

Prototype

```
virtual void AdjustCursor(
     const EventRecord &inMacEvent);
```

Parameters The parameter for this method is:

`const EventRecord&`	`inMacEvent`	This passed-in reference tells the location of the mouse cursor.

Return None

ClickMenuBar()

Purpose Respond to a mouse click in the menu bar.

Access Virtual, Protected

Prototype

```
virtual void ClickMenuBar(
     const EventRecord &inMacEvent);
```

Parameters The parameter for this method is:

`const EventRecord&`	`inMacEvent`	This passed-in reference contains event information.

Return None

DispatchEvent()

Purpose This method processes a Mac OS Toolbox event.

Access Virtual, Public

Prototype

```
virtual void DispatchEvent(
     const EventRecord &inMacEvent);
```

Parameters The parameter for this method is:

`const EventRecord&`	`inMacEvent`	This passed-in reference contains event information.

Return None

EventActivate()

Purpose Respond to an Activate (or Deactivate) event by enabling the appropriate window and updating the command status.

Access Virtual, Protected

Prototype

```
virtual void EventActivate(
     const EventRecord &inMacEvent);
```

Parameters The parameter for this method is:

`const EventRecord&`	`inMacEvent`	This passed-in reference contains event information.

Return None

EventAutoKey()

Purpose Respond to an auto-key event.

Access Virtual, Protected

Prototype

```
virtual void EventAutoKey(
     const EventRecord &inMacEvent);
```

Parameters The parameter for this method is:

`const EventRecord&`	`inMacEvent`	This passed-in reference contains event information.

Return None

EventDisk()

Purpose Respond to a Disk-inserted event.

Access Virtual, Protected

Prototype

```
virtual void EventDisk(
     const EventRecord &inMacEvent);
```

Parameters The parameter for this method is:

`const EventRecord&`	`inMacEvent`	This passed-in reference contains event information.

Return None

EventHighLevel()

Purpose Respond to a High Level (Apple) Event.

Access Virtual, Protected

Prototype

```
virtual void EventHighLevel(
     const EventRecord &inMacEvent);
```

Parameters The parameter for this method is:

`const EventRecord&`	`inMacEvent`	This passed-in reference contains event information.

Return None

EventKeyDown()

Purpose Respond to a Key Down Event.

Access Virtual, Protected

Prototype
```
virtual void EventKeyDown(
     const EventRecord &inMacEvent);
```

Parameters The parameter for this method is:

`const EventRecord&`	`inMacEvent`	This passed-in reference contains event information.

Return None

EventKeyUp()

Purpose Respond to a Key Up event. Note that by default, the system masks out Key Up events.

Access Virtual, Protected

Prototype
```
virtual void EventKeyUp(
     const EventRecord& inMacEvent);
```

Parameters The parameter for this method is:

`const EventRecord&`	`inMacEvent`	This passed-in reference contains event information.

Return None

EventMouseDown()

Purpose Respond to a Mouse Down event.

Access Virtual, Protected

Prototype
```
virtual void EventMouseDown(
     const EventRecord &inMacEvent);
```

Parameters The parameter for this method is:

`const EventRecord&`	`inMacEvent`	This passed-in reference contains event information.

Return None

EventMouseUp()

Purpose Respond to a Mouse Up event.

Access Virtual, Protected

Prototype

```
virtual void EventMouseUp(
     const EventRecord &inMacEvent);
```

Parameters The parameter for this method is:

`const EventRecord&`	`inMacEvent`	This passed-in reference contains event information.

Return None

EventOS()

Purpose Respond to an OS event (MouseMoved, Suspend, Resume).

Access Virtual, Protected

Prototype

```
virtual void EventOS(
     const EventRecord &inMacEvent);
```

Parameters The parameter for this method is:

`const EventRecord&`	`inMacEvent`	This passed-in reference contains event information.

Return None

EventResume()

Purpose Respond to a Resume event.

Access Virtual, Protected

Prototype

```
virtual void EventResume(
     const EventRecord& inMacEvent);
```

Parameters The parameter for this method is:

`const EventRecord&`	`inMacEvent`	This passed-in reference contains event information.

Return None

EventSuspend()

Purpose Respond to a Suspend event.

Access Virtual, Protected

Prototype

```
virtual void EventSuspend(
     const EventRecord& inMacEvent);
```

Parameters The parameter for this method is:

`const EventRecord&`	`inMacEvent`	This passed-in reference contains event information.

Return None

EventUpdate()

Purpose Respond to an Update event.

Access Virtual, Protected

Prototype

```
virtual void EventUpdate(
     const EventRecord &inMacEvent);
```

Parameters The parameter for this method is:

`const EventRecord&`	`inMacEvent`	This passed-in reference contains event information.

Return None

ExecuteAttachments()

Purpose Tell all associated Attachments to execute themselves for the specified message.

Access Virtual, Public

Prototype

```
virtual Boolean ExecuteAttachments(
     MessageT inMessage,
     void *ioParam);
```

Parameters The parameters for this method are:

`MessageT`	`inMessage`	This passed-in value indicates the message to process.
`void*`	`ioParam`	This passed-in pointer is for a data block that accompanies the `inMessage`.

Return The return value specifies whether the default Host action should be executed. The value is `false` if any Attachment's `Execute()` function returns `false`, otherwise it's `true`.

GetCurrentEventDispatcher()

Purpose Get a pointer to the current LEventDispatcher object so that it can be later restored.

Access Static, Public

Prototype
```
static LEventDispatcher
      *GetCurrentEventDispatcher();
```

Parameters None

Return A `LEventDispatcher` pointer that points to the current LEvent-Dispatcher.

Remarks This is a static method, and as such the same method will be called for every instance of the LEventDispatcher objects.

UpdateMenus()

Purpose This method handles all the menu update operations that are required for event processing.

Access Virtual, Public

Prototype
```
virtual void UpdateMenus();
```

Parameters None

Return None

Remarks This method handles many cases, and is best understood by reading the source code.

UseIdleTime()

Purpose Respond to a NULL or MouseMoved Event.

Access Virtual, Public

Prototype
```
virtual void UseIdleTime(
     const EventRecord &inMacEvent);
```

Parameters The parameter for this method is:

`const EventRecord&`	`inMacEvent`	This passed-in reference contains event information.

Return None

Remarks Typically, all this method needs to do is devote time to processing idle tasks.

sCurrentDispatcher

Purpose This data member provides a place to store the current EventDispatcher object pointer.

Access Protected

Prototype
```
static LEventDispatcher *sCurrentDispatcher;
```

mSaveDispatcher

Purpose This data member provides a place to store the previous EventDispatcher object pointer. From this, the former EventDispatcher can be restored later.

Access Protected

Prototype
```
LEventDispatcher *mSaveDispatcher;
```

mMouseRgnH

Purpose This data member provides a place to store the mouse region handle.

Access Protected

Prototype

```
RgnHandle mMouseRgnH;
```

LEventSemaphore

Overview LEventSemaphore is a PowerPlant class that overrides LSemaphore.

Methods The methods in this class are:

LEventSemaphore() ~LEventSemaphore()

Reset() Signal()

Data Members The data members in this class are:

mPostCount

Operation The Signal() method is overridden so that it can be made to simultaneously release all threads waiting on this semaphore. The Reset() method is guaranteed to raise the semaphore so that no thread can access the flagged data until the next call to Signal().

Source files (Thread Classes)

`LEventSemaphore.h`

`LEventSemaphore.cp`

See also LSemaphore

LEventSemaphore()

Purpose The constructor creates the object.

Access Public

Prototype

```
LEventSemaphore();
LEventSemaphore(Boolean posted);
```

Parameters A Boolean indicating whether the semaphore is posted or not. If posted is true, the semaphore is marked as being in the posted state, and any calls to Wait() will return immediately.

~LEventSemaphore()

Purpose The destructor destroys the object.

Access Virtual, Public

Prototype
```
~LEventSemaphore();
```

Reset()

Purpose Guaranteed to raise the semaphore so that no thread can access the flagged data until the next call to Signal(). Throws an exception if the semaphore is already reset.

Access Virtual, Public

Prototype
```
virtual Uint32 Reset();
```

Parameters None

Return Returns the number of times that Signal() was called since the last call to Reset().

Signal()

Purpose Make the semaphore available to all threads. This includes all blocked threads, as well as those that will call Wait() before the next call to Reset(). This is an override of Signal() in LSemaphore.

Access Virtual

Prototype
```
virtual void Signal();
```

Parameters None

Return None

mPostCount

Purpose The "post" counter

Access protected

Prototype

```
Uint32 mPostCount;
```

LFocusBox

Overview LFocusBox is a PowerPlant class that is used for outlining a pane to indicate that the pane is the current focus for keystrokes.

Methods The methods in this class are:

LFocusBox()	AttachPane()
DontRefresh()	DrawSelf()
GetBoxRegion()	HideSelf()
Refresh()	ShowSelf()

Data Members There are no data members in this class.

Operation This class is used internally by PowerPlant in conjunction with LListBox to highlight entries in the list.

Source files (Pane Classes)

`LFocusBox.h`

`LFocusBox.cp`

Ancestors LAttachable

LPane

LFocusBox()

Purpose The constructors create objects from the passed-in parameters.

Access Public

Prototype

```
LFocusBox();
LFocusBox( const LFocusBox &inOriginal );
LFocusBox( LStream *inStream );
```

Parameters The parameters for these constructors are:

`const LFocusBox&`	`inOriginal`	The reference to the object to copy.
`LStream*`	`inStream`	The stream to read from.

AttachPane()

Purpose Associate a host Pane with a FocusBox, sizing and positioning the FocusBox so that it fits around the Pane.

Access Virtual, Public

Prototype

```
virtual void AttachPane(
    LPane *inPane,
    Boolean inSameBindings );
```

Parameters The parameters for this method are:

`LPane*`	`inPane`	The pointer to the pane.
`Boolean`	`inSame-Bindings`	If true, FocusBox uses same frame bindings as the host Pane. This makes the FocusBox move and resize along with the host Pane if their mutual SuperView moves or resizes.

Return None

DontRefresh()

Purpose Validate the area occupied by a FocusBox so it won't redraw during the next Update event. This is an override of DontRefresh() in LView.

DrawSelf()

Purpose Draw a FocusBox. This is an override of DrawSelf() in LPane.

GetBoxRegion()

Purpose Pass back a region defining a FocusBox. A FocusBox is a hollow rectangular region. Clip the region so that it is within the revealed rect. This method creates a new region which the caller must dispose of.

Access Protected

Prototype

```
virtual RgnHandle GetBoxRegion(
     const Rect &inFrame,
     const Rect &inRevealed ) const;
```

Parameters The parameters for this method are:

`const Rect&`	`inFrame`	Reference to the frame Rect.
`const Rect&`	`inRevealed`	Reference to the revealed Rect.

Return A handle to the region defining the FocusBox.

HideSelf()

Purpose A FocusBox is being hidden. This is an override of HideSelf() in LPane.

Refresh()

Purpose Invalidate the area occupied by a FocusBox so it will redraw during the next Update event. This is an override of Refresh() in LPane.

ShowSelf()

Purpose A FocusBox is being shown. This is an override of ShowSelf() in LPane.

LGlobalsContext

Overview

`LGlobalsContext` is a PowerPlant class that is a mix-in for any class which needs to save the A5 or A4 (global variables) context for use in a Toolbox callback.

Methods

The methods in this class are:

LGlobalsContext() ~LGlobalsContext()

GetGlobals()

Data Members

The data members in this class are:

mSavedGlobals

Operation

This class is defined as a null class for PowerPC or 68K CFM builds. Designed to be used in concert with StSetupGlobals.

Source files

(Networking Classes)

`UCallbackUtils.h`

`UCallbackUtils.cp`

See also

LMacTCPDNSOperation

LMacTCPTCPEndpoint

LMacTCPUDPEndpoint

StMacTCPOperation

StMacTCPUDPOperation

StOpenTptOperation

LGlobalsContext()

Purpose The constructor creates the object, saving global variables context. Assumes that the global variables register (either A5 or A4) is correct at this time. On PowerPC or 68K CFM, it does nothing.

Access Public

Prototype
```
inline LGlobalsContext();
```

Parameters None

~LGlobalsContext()

Purpose The destructor destroys the object.

Access Public

Prototype
```
inline ~LGlobalsContext();
```

GetGlobals()

Purpose Returns the value of mSavedGlobals.

Access Public

Prototype
```
inline Int32 GetGlobals();
```

Parameters None

Return The value of mSavedGlobals.

mSavedGlobals

Purpose The saved globals in A5 or A4.

Access Private

Prototype `Int32 mSavedGlobals;`

LGroupBox

Overview LGroupBox is a PowerPlant class that is usually used for visually grouping user interface elements, using a line drawn around them.

Methods The methods in this class are:

LGroupBox()	~LGroupBox()
DrawSelf()	DrawText()
DrawBorder()	CalcTextBoxFrame()

Data Members The data members in this class are:

mFrameColor

Operation This class derives from LCaption. The text set with DrawText() is the title of the group. This object draws a box with the specified co-ordinates.

The top of the object's frame does not coincide with the top line drawn for the group box. The other panes that are visually within the group box do not have a programming relationship to the group box in the hierarchy. They are simply visually grouped, without any program control between the group box and the other panes.

Source files (Pane Classes)

`LGroupBox.h`

`LGroupBox.cp`

See also LCaption

LGroupBox()

Purpose The constructor creates the object from the passed-in parameters.

Access Public

Prototype

```
LGroupBox();
LGroupBox( const LGroupBox &inGroupBox );
LGroupBox( const SPaneInfo &inPaneInfo,
     Str255 inString,
     ResIDT inTextTraitsID );
LGroupBox( LStream *inStream );
```

Parameters The parameters for these constructors are:

`const LGroupBox&`	`inGroupBox`	A reference to the group box object to be copied.
`const SPaneInfo&`	`inPaneInfo`	A reference to the SuperPane to draw the group box upon.
`Str255`	`inString`	A pointer to the group box text.
`ResIDT`	`inTextTraitsID`	The text traits for the group box text.
`LStream*`	`inStream`	A pointer to a stream object that contains the information to create the LCaption object.

~LGroupBox()

Purpose The destructor destroys the object.

Access Virtual, Public

Prototype

```
virtual LGroupBox();
```

Parameters None

DrawSelf()

Purpose This method draws the group box. It is drawn in two stages. First the frame rectangle is drawn, minus the area where the title text is drawn. Second, the title text is drawn. In order to draw without any flashing, we find the frame of the text box and clip it out of the drawing region.

Access Virtual, Protected

Prototype
```
virtual void DrawSelf();
```

Parameters None

Return None

DrawText()

Purpose This method draws the group box text given the chosen `Rect` coordinates.

Access Virtual, Protected

Prototype
```
virtual void DrawText( const Rect &inRect );
```

Parameters The parameter for this method is:

`const Rect& inRect`	A reference to the `Rect` to draw the group box text in.

Return None

DrawBorder()

Purpose This method draws the group box frame given the chosen coordinates.

Access Virtual, Protected

Prototype
```
virtual void DrawBorder( const Rect &inRect );
```

Parameters The parameter for this method is:

`const Rect& inRect`	A reference to the `Rect` to draw the group box upon.

Return None

CalcTextBoxFrame()

Purpose This method decides where the text will be drawn, but does not draw the text.

Access Virtual, Protected

Prototype

```
virtual void CalcTextBoxFrame( Rect &outRect );
```

Parameters The parameter for this method is:

`const Rect&  outRect`	A reference to the `Rect` to draw the group box text within.

Return None

mFrameColor

Purpose This data member holds the color to draw the group box frame in.

Access Protected

Prototype

```
RGBColor mFrameColor;
```

LGrowZone

Overview

LGrowZone is a PowerPlant class to use for managing any low memory situations that occur during your program's execution. For general information about the Mac OS GrowZone concepts, refer to *Inside Macintosh: Memory*, published by Addison-Wesley.

LGrowZone's implementation provides two levels of functionality. First, it provides the classic behavior of allocating a user-specified emergency reserve buffer of memory upon creation of the GrowZone. The OS will then notify your application when to draw from this emergency reserve in low-memory situations. Second, LGrowZone can work with other PowerPlant objects to ask them to free memory when it becomes scarce.

Methods

The methods in the LGrowZone class are:

LGrowZone()	~LGrowZone()
AskListenersToFree()	DoGrowZone()
GetGrowZone()	GiveWarning()
GrowZoneCallBack()	MemoryIsLow()
SpendTime()	UseLocalReserve()

Data Members

The data members in the LGrowZone class are:

sGrowZone	sGrowZoneUPP
mLocalReserve	mReserveSize
mGiveWarning	

Operation

Near the start of your application code you should create a single instance of LGrowZone, as shown here:

```
main() {
  // do init stuff here

  // create a memory reserve of 32K bytes
  new LGrowZone( 32 * 1024 );
```

```
  // the rest of your code goes here...
}
```

LGrowZone inherits from LBroadcaster. Objects which are able to free memory when it is becoming depleted should be Listeners, and attach themselves to the LGrowZone object so they will be notified when memory is low. The following code shows how to add your objects as listeners for this low-memory notification. This code attaches `myObj` as a Listener to the LGrowZone messages.

```
LGrowZone::GetGrowZone()->AddListener(myObj);
```

When memory is needed, you'll want `myObj` to be able to respond to a ListenToMessage() call. The call to `ListenToMessage()` will occur with a `msg_GrowZone` message, and a pointer to the number of bytes needed. You should add code similar to the following to your object's class definition to handle the `ListenToMessage()` method, that you inherit from LBroadcaster.

```
MyClass::ListenToMessage(MessageT inMessage,
                         void *ioParam)
{
    if (inMessage == msg_GrowZone) {

        // Memory is low, free our cache
        Int32 freedBytes =
        ::GetHandleSize(myCache);
        ::DisposeHandle(myCache);
        myCache = nil;

        // Pass back bytes freed
        *(Int32*)ioParam = freedBytes;
    }
}
```

For example, suppose `MyClass` stores the `myCache` Handle with data generated from a lengthy calculation. If memory gets low, it can dispose of the cached data, meaning that it will have to recalcu-

late the data if it needs it later. This is a common trade-off: memory versus speed.

This class also inherits from LPeriodical. LGrowZone's constructor installs the object in the Periodical Repeater queue so that its SpendTime() function gets called each time through the main event loop. This is required so that the LGrowZone object can monitor memory usage and can respond appropriately, perhaps by displaying a warning to the user.

Source files (Support Classes)

`LGrowZone.h`

`LGrowZone.cp`

See also LBroadcaster

LPeriodical

LGrowZone()

Purpose The constructor allocates a memory reserve of the passed-in size, and a GrowZone function for the current process is then registered with the operating system.

The constructor also installs this LGrowZone object in the Periodical Repeater queue defined by LPeriodical, so that the SpendTime() method will get called each time through the application's main event loop.

Access Public

Prototype

```
LGrowZone(Size inReserveSize);
```

Parameters This method has the following parameter:

`Size`	`inReserveSize`	Indicates the size of the memory block that you wish to reserve in your heap area. The unit of this value is in bytes.

Return None

~LGrowZone()

Purpose The destructor destroys the LGrowZone object. This includes deallocating the previously-allocated memory reserve and deinstalling the GrowZone function.

Access Public

Prototype
```
virtual ~LGrowZone();
```

Parameters None

Return None

AskListenersToFree()

Purpose This method broadcasts a message to listeners that they should free up some memory if possible. In order for an object to be a listener, it must have registered with the LGrowZone object using AddListener().

Access Virtual, Public

Prototype
```
virtual Int32 AskListenersToFree(
     Size inBytesNeeded );
```

Parameters This method has the following parameter:

`Size`	`inBytesNeeded`	The number of bytes needed.

Return `Int32` containing the number of bytes that were freed as a result of calling this method. If no memory could be freed, then it returns 0.

Remarks

- This method sends a `msg_GrowZone` message (via the ListenToMessage() call) to each object that has registered as a Listener, with a parameter that is a pointer to the number of bytes needed. On exit, objects that are Listeners should set this pointer equal to the number of bytes that they deallocated.
- A running count of the bytes needed is updated, which could be negative if some Listener frees more than we need.

DoGrowZone()

Purpose This method is called by the GrowZone function installed by the LGrowZone() constructor when the operating system needs more memory.

Access Virtual, Protected

Prototype

```
virtual DoGrowZone( Size inBytesNeeded );
```

Parameters This method has the following parameter:

`Size` `inBytesNeeded` The number of bytes needed.

Return `Int32` containing the number of bytes that were freed as a result of calling this method. If no memory could be freed, then it returns 0.

Remarks This method is not normally called directly in your normal application code.

GetGrowZone()

Purpose This method is an accessor, designed to return a pointer to the LGrowZone object.

Access Inline, Public, Static

Prototype `static LGrowZone* GetGrowZone();`

Parameters None

Return A pointer to the LGrowZone object in your application.

Remarks You might need this method if you are going to call AddListener(). This would be done to register an object for future notification of low-memory conditions.

GiveWarning()

Purpose This method sets the mGiveWarning data member to true.

Access Public, Inline

Prototype `void GiveWarning();`

Parameters None

Return None

Remarks You might want to call this method if you wish to have the PowerPlant framework display a generic alert to the user when memory is depleted.

GrowZoneCallBack()

Purpose This method is called when the System requires more memory.

Access Protected, Static

Prototype
```
static pascal Int32 GrowZoneCallBack(
     Size inBytesNeeded );
```

Parameters This method has the following parameter:

`Size inBytesNeeded` The number of bytes needed.

Return Int32 containing the number of bytes that were freed by your program.

Remarks This is the GrowZone function registered with the System. It sets up the Motorola 68K A5 world so we can access globals, then calls a virtual function for the LGrowZone class.

MemoryIsLow()

Purpose This method provides a way of determining whether the memory reserve allocated by the LGrowZone() constructor has been utilized.

Access Virtual, Public

Prototype
```
virtual Boolean MemoryIsLow();
```

Parameters None

Return Return true if the LGrowZone() memory reserve has been used, false otherwise.

Remarks Objects in your program can call this method if they wish to behave differently under low memory situations. For example, a program could disable the "New" and "Open" commands to prevent new Documents from being created when memory is low.

SpendTime()

Purpose This method performs periodic maintenance for the LGrowZone object, including

Access Virtual, Public

Prototype
```
virtual void SpendTime(
     const EventRecord& inMacEvent );
```

Parameters This method has the following parameter:

`const EventRecord&`	`inMacEvent`	a struct containing information about the Mac OS event that has just occurred

An `EventRecord` reference is the parameter to this method, but it is not utilized by this override.

Return None

Remarks

- This method is an override of the SpendTime() base class method from LPeriodical().
- This method attempts to reallocate the LGrowZone() memory reserve if necessary, and (optionally) warn the user if memory is getting low.
- This method is called each time through the main event loop.
- In order for the framework to provide a warning, the GiveWarning() method must be called first.

UseLocalReserve()

Purpose This method empties the reserve that was allocated during creation of the LGrowZone object.

Access Virtual, Protected

Prototype

```
virtual Int32 UseLocalReserve(Size inBytesNeeded );
```

Parameters This method has the following parameter:

`Size`	`inBytesNeeded`	The number of bytes needed.

Return Int32 containing the number of bytes that were freed.

Remarks

- Calling this method is a last chance desperate attempt to free enough memory to proceed.
- If the memory reserve for LGrowZone exists, it will be freed. Also, mGiveWarning is set to true. The user will receive

warning alerts from the framework if memory becomes low again.

sGrowZone

Purpose This is a pointer to the LGrowZone object. It is accessible using the GetGrowZone() accessor method.

Access Static, Protected

Prototype
```
static LGrowZone* sGrowZone;
```

Remarks You do not normally need to directly access this data member, and should use the GetGrowZone() accessor method instead.

sGrowZoneUPP

Purpose This is a Universal Procedure Pointer (UPP) used for pointing to the GrowZone procedure.

Access Static, Protected

Prototype
```
static GrowZoneUPP sGrowZoneUPP;
```

mLocalReserve

Purpose This member is a Handle to the memory reserve allocated by the LGrowZone() constructor.

Access Protected

Prototype
```
Handle mLocalReserve;
```

mReserveSize

Purpose This member is a count of the size (in bytes) of the memory reserve area allocated by the LGrowZone() constructor.

Access Protected

Prototype
```
Size mReserveSize;
```

mGiveWarning

Purpose This data member is set to true if you want the PowerPlant framework to provide a default warning alert to the user under low-memory conditions.

Access Protected

Prototype
```
Boolean GiveWarning;
```

Remarks The default for this data member is false when the LGrowZone() constructor is invoked.

LGWorld

Description LGWorld is a PowerPlant class that is used for creating GWorlds for offscreen drawing.

Methods The methods in this class are:

LGWorld()	~LGWorld()
BeginDrawing()	CopyImage()
EndDrawing()	GetBounds()
GetMacGWorld()	SetBounds()

Data Members The data members in this class are:

mMacGWorld	mBounds
mSavePort	mSaveDevice

Source files (Pane Classes)

`UGWorld.h`

`UGWorld.cp`

Ancestors You can use LGWorld to help manage the creation and use of off-screen drawing areas. Refer to *The PowerPlant Book* for an example on how to use this class.

LGWorld()

Purpose The constructor creates an object from the passed-in parameters.

Access Public

Prototype

```
LGWorld( const Rect &inBounds,
     Int16 inPixelDepth = 0,
     GWorldFlags inFlags = 0 ,
```

```
    CTabHandle inCTableH = nil,
    GDHandle inGDeviceH = nil );
```

Parameters The parameters for this constructor are:

`const Rect&`	`inBounds`	The bounds Rect for the drawing area.
`Int16`	`inPixelDepth`	The pixel depth, the default is 0.
`GWorldFlags`	`inFlags`	The flags, the default is 0.
`CTabHandle`	`inCTableH`	Handle to a color table, default is `nil`.
`GDHandle`	`inGDeviceH`	Handle to the GDevice, default is `nil`.

~LGWorld()

Purpose The destructor destroys the object.

Access Virtual, Public

Prototype

```
virtual ~LGWorld();
```

BeginDrawing()

Purpose Sets the current port to the offscreen GWorld and locks its pixels. Every `BeginDrawing` call must be balanced by a corresponding EndDrawing() call.

Access Public

Prototype

```
Boolean BeginDrawing();
```

Parameters None

Return Returns `false` if the offscreen pixels can't be locked. This will happen if the `pixPurge` flag was set and the pixels were purged from memory. If so, you might want to try to reallocate the pixels by calling the Toolbox routine `UpdateGWorld()`.

CopyImage()

Purpose Copies an image from the offscreen GWorld to the specified port.

Access Public

Prototype

```
void CopyImage(
     GrafPtr inDestPort,
     const Rect &inDestRect,
     Int16 inXferMode,
     RgnHandle inMaskRgn );
```

Parameters The parameters for this constructor are:

`GrafPtr`	`inDestPort`	The destination port.
`const Rect&`	`inDestRect`	The destination port Rect.
`Int16`	`inXferMode`	The transfer mode, see `QuickDraw.h`.
`RgnHandle`	`inMaskRgn`	The handle to the mask region.

Return None

Remarks You will want to make sure that the foreground color is black and the background color is white before calling this function. `CopyBits()` can be unreliable if this is not the case. This isn't done automatically here ao that this routine to be fast (that's why you usually use offscreen drawing).

EndDrawing()

Purpose Unlocks the GWorld's pixels and restores the current port to what it was before BeginDrawing() was called. Every EndDrawing call must be preceded by a corresponding `BeginDrawing()` call.

Access Public

Prototype
```
void EndDrawing();
```

Parameters None

Return None

GetBounds()

Purpose Returns the value of the mBounds data member in the passed-in parameter.

Access Inline, Public

Prototype
```
void GetBounds( Rect &outBounds );
```

Parameters The parameters for this method is:

`Rect&`	`outBounds`	The bounds Rect.

Return None

GetMacGWorld()

Purpose This method retrieves a pointer to a Mac OS GWorld.

Access Inline, Public

Prototype
```
GWorldPtr GetMacGWorld();
```

Parameters None

Return Pointer to a Mac OS GWorld.

SetBounds()

Purpose Used to adjust the bounds of an offscreen world.

Access Public

Prototype
```
void SetBounds( const Rect &inBounds );
```

Parameters The parameters for this method is:

`Rect&`	`outBounds`	The bounds Rect.

Return None

mMacGWorld

Purpose Storage for the Mac OS Gworld.

Access Protected

Prototype
```
GWorldPtr mMacGWorld;
```

mBounds

Purpose Storage for the bounds Rect.

Access Protected

Prototype
```
Rect mBounds;
```

mSavePort

Purpose Storage to save the port.

Access Protected

Prototype `CGrafPtr mSavePort;`

mSaveDevice

Purpose Storage to save the handle to the device.

Access Protected

Prototype `GDHandle mSaveDevice;`

LHandleStream

Overview LHandleStream is a PowerPlant class that is used for doing LStream operations with data blocks that are relocatable, such as handles.

Methods The methods in this class are:

LHandleStream()	~LHandleStream()
DetachDataHandle()	GetBytes()
GetDataHandle()	PutBytes()
SetDataHandle()	SetLength()

Data Members The data members in this class are:

mDataH

Operation General usage for LHandleStream is similar to that of LDataStream and LFileStream.

Source files File & Stream Classes)

`LHandleStream.h`

`LHandleStream.cp`

Ancestors LStream

LHandleStream()

Purpose The constructor creates objects.

Access Public

Prototype

```
LHandleStream();
LHandleStream( Handle inHandle);
```

Parameters The parameter for the constructors is:

`Handle`	`inHandle`	The handle to create an object from.

~LHandleStream()

Purpose The destructor destroys the object.

Access Virtual, Public

Prototype

```
virtual ~LHandleStream();
```

DetachDataHandle()

Purpose Dissociate the data Handle from a HandleStream. This creates a new, empty data Handle and passes back the existing Handle. Caller assumes ownership of the Handle.

Access Public

Prototype

```
Handle DetachDataHandle();
```

Parameters None

Return A new, empty data handle.

GetBytes()

Purpose Read bytes from a HandleStream to a buffer.

Access Virtual, Public

Prototype

```
virtual ExceptionCode GetBytes(
     void *outBuffer,
     Int32 &ioByteCount );
```

Parameters The parameters for this method are:

`void*`	`outBuffer`	The pointer to a buffer to read into.
`Int32&`	`ioByteCount`	The size of the buffer on input, the number of bytes read on output.

Return Returns an error code and passes back the number of bytes actually read, which may be less than the number requested if an error occurred.

- readErr—Attempt to read past the end of the HandleStream

GetDataHandle()

Purpose Retrieves the value of mDataH.

Access Public

Prototype
```
Handle GetDataHandle();
```

Parameters None

Return Returns the value of mDataH.

PutBytes()

Purpose Write bytes from a buffer to a HandleStream. Grows the data Handle if necessary.

Access Public

Prototype
```
virtual ExceptionCode PutBytes(
     const void *inBuffer,
     Int32 &ioByteCount );
```

Parameters The parameters for this method are:

`const void*`	`inBuffer`	The pointer to a buffer to write into.
`Int32&`	`ioByteCount`	The size of the buffer on input, the number of bytes written on output.

Return Returns an error code and passes back the number of bytes actually written, which may be less than the number requested if an error occurred.

- `memFullErr`—Growing Handle failed when trying to write past the current end of the Stream

SetDataHandle()

Purpose Specify a Handle to use as the basis for a HandleStream.

Class assumes ownership of the input Handle and destroys the existing data Handle. Call DetachDataHandle() beforehand if you wish to preserve the existing data Handle.

Access Public

Prototype
```
void SetDataHandle( Handle inHandle );
```

Parameters The parameter for this method is:

`Handle`	`inHandle`	The handle to set for the object.

Return None

SetLength()

Purpose Set the length, in bytes, of the HandleStream.

Access Public

Prototype `virtual void SetLength( Int32 inLength );`

Parameters The parameter for this method is:

`Int32`	`inLength`	The length of the stream.

Return None

mDataH

Purpose The data handle.

Access Protected

Prototype `Handle mDataH;`

LIconPane

Overview

LIconPane is a PowerPlant class that is used for drawing a single icon from an icon family.

Methods

The methods in this class are:

LIconPane() ~LIconPane()

DrawSelf() SetIconID()

Data Members

The data members in this class are:

mIconID

Operation

Because this class uses the Mac OS icon-handling routines, an LIconPane object draws the member of the icon family that best fits the color settings and pixel depth of the current device.

Source files

(Pane Classes)

`LIconPane.h`

`LIconPane.cp`

Ancestors

LAttachable

LPane

LIconPane()

Purpose

The constructors create objects from the passed-in parameters.

Access

Public

Prototype

```
LIconPane();
LIconPane( const SPaneInfo &inPaneInfo,
     ResIDT inIconID );
LIconPane( LStream *inStream );
```

Parameters These constructors have the following parameters:

`const SPaneInfo&`	`inPaneInfo`	The reference to the Superpane.
`ResIDT`	`inIconID`	The resource ID for the icon.
`LStream*`	`inStream`	The stream to construct an object from.

~LIconPane()

Purpose The destructor destroys the object.

Access Virtual, Public

Prototype
```
virtual ~LIconPane();
```

DrawSelf()

Purpose Draws the icon. This is an override of DrawSelf() in LPane.

Access Public

Prototype
```
virtual void DrawSelf();
```

Parameters None

Return None

SetIconID()

Purpose Set the icon.

Access Public

Prototype
```
void SetIconID( ResIDT inIconID );
```

Parameters These constructors have the following parameters:

`ResIDT`	`inIconID`	The resource ID for the icon.

Return None

mIconID

Purpose Storage for the icon resource ID.

Access Protected

Prototype

```
ResIDT mIconID;
```

LInternetAddress

Overview

`LInternetAddress` is a PowerPlant class that is used for wrapping the IP addresses that are used internally by the Internet to describe another computer's location.

Methods

The methods in this class are:

LInternetAddress()	~LInternetAddress()
Clone()	GetDNSAddress()
GetDNSDescriptor()	GetHostPort()
GetIPAddress()	GetIPAddress()
GetIPDescriptor()	InternalLookupAddress()
InternalLookupName()	MakeOTDNSAddress()
MakeOTIPAddress()	SetDNSAddress()
SetHostPort()	SetIPAddress()

Data Members

The data members in this class are:

mIPAddress	mDNSAddress
mHostPort	

Operation

DNS names (like "ftp.metrowerks.com") must be mapped to IP addresses (127.0.0.1) before connections are established. This object will handle such conversions automatically.

Source files

(Networking Classes)

`LInternetAddress.h`

`LInternetAddress.cp`

LInternetAddress()

Purpose

The constructor creates the object.

Access Public

Prototype

```
LInternetAddress(
     Uint32 inHostAddress,
     Uint16 inHostPort );

LInternetAddress(
     ConstStringPtr inHostAddress,
     Uint16 inHostPort,
     Boolean inLookupNow );

LInternetAddress(
     Uint16inHostPort );

LInternetAddress(
     const LInternetAddress& inOriginal );
```

Parameters The parameters for this method are:

`Uint32`	`inHostAddress`	The host address.
`Uint16`	`inHostPort`	The host port.
`ConstStringPtr`	`inHostAddress`	The host address.
`Boolean`	`inLookupNow`	Whether to do an immediate lookup.
`LInternetAddress&`	`inOriginal`	The address to copy.

~LInternetAddress()

Purpose The destructor destroys the `LInternetAddress` object.

Access Virtual, Public

Prototype

```
virtual ~LInternetAddress();
```

Clone()

Purpose Create a new copy of the address using the `new` operator from this one.

Access Virtual, Public

Prototype
```
virtual LInternetAddress* Clone();
```

Parameters None

Return A new copy of the internet address.

GetDNSAddress()

Purpose Retrieve the DNS address string.

Access Virtual, Public

Prototype
```
virtual StringPtr GetDNSAddress(
     Str255outDescriptor);
```

Parameters The parameters for this method are:

`Str255`	`outDescriptor`	The address string.

Return Returns the address string.

GetDNSDescriptor()

Purpose Return the name of the specified host (and port number if available) Use GetDNSAddress() if you don't care if we have tried to do a DNS lookup.

Access Virtual, Public

Prototype
```
StringPtr GetDNSDescriptor( Str255outDescriptor,
Boolean withPort);
```

Parameters The parameters for this method are:

`Str255`	`outDescriptor`	The address string.
`Boolean`	`withPort`	Whether to add the port to the string or not.

Return A pointer to the descriptor.

GetHostPort()

Purpose Returns the value of mHostPort.

Access Virtual, Public

Prototype
```
virtual Uint16 GetHostPort();
```

Parameters None

Return The value of mHostPort.

GetIPAddress()

Purpose Return the 32-bit IP address of the host.

Access Virtual, Public

Prototype
```
virtual Uint32 GetIPAddress();
```

Parameters None

Return The 32-bit address.

GetIPAddress()

Purpose Return the 32-bit IP address of the host as well as dotted decimal format.

Access Virtual, Public

Prototype
```
virtual Uint32 GetIPAddress(Str255outDescriptor);
```

Parameters The parameters for this method are:

`Str255`	`outDescriptor`	The address string.

Return The 32-bit address.

GetIPDescriptor()

Purpose Convert the IP address to dotted decimal format and return this string. If a port number is specified, optionally return that at the end of the string.

Access Virtual, Public

Prototype
```
virtual StringPtr GetIPDescriptor(
    Str255outDescriptor, Boolean withPort);
```

Parameters The parameters for this method are:

`Str255`	`outDescriptor`	The address string.
`Boolean`	`withPort`	Add the port to the string or not.

Return A string pointer.

InternalLookupAddress()

Purpose Create mapper and lookup address from DNS name.

Access Private

Prototype
```
void InternalLookupAddress();
```

Parameters None

Return None

InternalLookupName()

Purpose Create mapper and lookup name from IP Address.

Access Private

Prototype

```
void InternalLookupName();
```

Parameters None

Return None

MakeOTDNSAddress()

Purpose Create an Open Transport (OT) DNS address.

Access Virtual, Public

Prototype

```
virtual void MakeOTDNSAddress( TNetbuf& outAddress
);
```

Parameters The parameters for this method are:

`TNetbuf&`	`outAddress`	The OT address.

Return None

MakeOTIPAddress()

Purpose Make an Open Transport (OT) IP address.

Access Virtual, Public

Prototype

```
virtual void MakeOTIPAddress(
     TNetbuf& outAddress );
```

Parameters The parameters for this method are:

`TNetbuf&`	`outAddress`	The OT address.

Return None

SetDNSAddress()

Purpose Set the value of the mDNSAddress data member.

Access Virtual, Public

Prototype

```
virtual void SetDNSAddress(
     ConstStringPtrinHostAddress);
```

Parameters The parameters for this method are:

`ConstStringPtr`	`inHostAddress`	The DNS address.

Return None

SetHostPort()

Purpose Set the value of the mHostPort data member.

Access Virtual, Public

Prototype

```
virtual void SetHostPort(Uint16 inHostPort);
```

Parameters The parameters for this method are:

`Uint16`	`inHostPort`	The host port.

Return None

SetIPAddress()

Purpose Set the value of the mIPAddress data member.

Access Virtual, Public

Prototype
```
virtual void SetIPAddress(Uint32 inHostAddress);
```

Parameters The parameters for this method are:

`Uint32`	`inHostAddress`	The IP address.

Return None

mIPAddress

Purpose The IP address.

Access Protected

Prototype
```
Uint32 mIPAddress;
```

mDNSAddress

Purpose The DNS address.

Access Protected

Prototype
```
LStr255 mDNSAddress;
```

mHostPort

Purpose The host port.

Access Protected

Prototype
```
Uint16 mHostPort;
```

LInternetMapper

Overview

LInternetMapper encapsulates the services of a Domain Name System server (DNS) for remote computers.

Methods

The methods in this class are:

LInternetMapper()	~LInternetMapper()
AbortThreadOperation()	AddressToName()
GetLocalAddress()	NameToAddress()

Data Members

There are no data members in this class.

Operation

You use an LInternetMapper object to locate the address of another computer on the network. You can convert from a DNS address (like www.metrowerks.com) to an IP address (such as 127.0.0.1) and vice-versa. All of these methods are pure virtual, forcing you to provide your own concrete implementations if you choose to not use LMacTCPInetMapper or LOpenTptInetMapper.

NOTE: For most connection-oriented applications, it is not necessary to use the mapper interface. The LEndpoint interface can accept LInternetDNSAddress objects in its Connect member function, which causes a name lookup to take place automatically.

LInternetMapper is an abstract base class (thus its constructor is declared protected). Use one of the predefined subclasses, such as LMacTCPInetMapper or LOpenTptInetMapper, or use the UNetworkFactory::CreateInternetMapper function to create the appropriate mapper for the system software that's installed on the user's machine.

Source files

(Networking Classes)

`LInternetMapper.h`

`LInternetMapper.cp`

See also

LEndpoint

LInternetDNSAddress

LMacTCPInetMapper

LOpenTptInetMapper

UNetworkFactory

LInternetMapper()

Purpose The copy constructor creates the object. Don't use the defaule constructor.

Access Private

Prototype
```
LInternetMapper(LInternetMapper&);
```

Parameters None

~LInternetMapper()

Purpose The destructor destroys the object.

Access Public, Virtual

Prototype
```
~LInternetMapper();
```

AbortThreadOperation()

Purpose Aborts a thread operation.

Access Pure virtual, Public

Prototype
```
virtual void AbortThreadOperation(
     LThread * inThread) = 0;
```

Parameters The parameters for this method are:

`LThread*`	`inThread`	The thread.

Return None

AddressToName()

Purpose Convert an address to a name.

Access Pure virtual, Public

Prototype

```
virtual void AddressToName(
     Uint32 inHostIP, LStr255& outHostName) = 0;
```

Parameters The parameters for this method are:

`Uint32`	`inHostIP`	The host IP address.
`LStr255&`	`outHostName`	The host name.

Return None

GetLocalAddress()

Purpose Obtain the local address.

Access Pure virtual, Public

Prototype

```
virtual LInternetAddress* GetLocalAddress() = 0;
```

Parameters None

Return A pointer to the address.

NameToAddress()

Purpose Convert a name to an address.

Access Pure virtual, Public

Prototype

```
Uint32 NameToAddress(
     ConstStringPtr inHostName ) = 0;
```

Parameters The parameters for this method are:

`ConstStringPtr`	`inHostName`	The host name.

Return None

LInterruptSafeList

Overview LInterruptSafeList is a PowerPlant class that is used for....

Methods The methods in this class are:

LInterruptSafeList() ~LInterruptSafeList()

Append() IsEmpty()

Remove()

Data Members The data members in this class are:

mQueue mIteratorQueue

Operation This class implements a container class which can add and remove members at interrupt time. The current implementation uses the Mac OS Toolbox calls `Enqueue()` and `Dequeue()`, but this may change at a later time.

Source files (Networking Classes)

`LInterruptSafeList.h`

`LInterruptSafeList.cp`

See also LDNSOperationDeleteQueue

LNetMessageQueue

LTCPParamBlockDeleteQueue

LUDPParamBlockDeleteQueue

LInterruptSafeList()

Purpose The constructor creates the object.

Access Public

Prototype `LInterruptSafeList();`

Parameters None

~LInterruptSafeList()

Purpose The destructor destroys the object.

Access Virtual, Public

Prototype
```
virtual ~LInterruptSafeList();
```

Append()

Purpose Call to add an item to the end of a list. Note that there is currently no way to insert an item at any other position in a list.

This method may be called at interrupt time.

Access Virtual, Public

Prototype
```
virtual void Append(
      LInterruptSafeListMember* inItem );
```

Parameters The parameter for this method is:

`LInterruptSafeListMember*`	`inItem`	The pointer to the list item.

Return None

IsEmpty()

Purpose Returns true if there are no entries in this list.

Access Virtual, Public

Prototype
```
virtual Boolean IsEmpty() const;
```

Parameters None

Return Return true if list is empty.

Remove()

Purpose Call to remove an item from a list.

This method may be called at interrupt time.

Access Virtual, Public

Prototype

```
virtual Boolean Remove(
    LInterruptSafeListMember* inItem);
```

Parameters The parameter for this method is:

`LInterruptSafeListMember*`	`inItem`	`The pointer to the list item.`

Return Returns false if unable to remove the item from the list (i.e. it was not a member of the list, or another thread of execution removed the item first).

mQueue

Purpose The queue head.

Access Private

Prototype

```
QHdr mQueue;
```

mIteratorQueue

Purpose An iterator for the queue.

Access Private

Prototype `QHdr mIteratorQueue;`

LKeyScrollAttachment

Overview

LKeyScrollAttachment is a PowerPlant class that is used for handling scrolling of a View using the `Home`, `End`, `PageUp`, or `PageDown` keyboard navigation keys.

Methods

The methods in this class are:

LKeyScrollAttachment() ExecuteSelf()

Data Members

The data members in this class are:

mViewToScroll

Operation

This class is for use only with `msg_KeyPress`.If you have a View that is also a Commander, you can attach a LKeyScrollAttachment to it to implement keyboard navigation.

If your View is not a Commander, but you still want to implement keyboard navigation, you can attach a LKeyScrollAttachment to a SuperView that is a Commander (such as the Window containing the View). However, if you can delete the View independent of the Window, you must take care to delete the Attachment.

Source files

(Utility Classes)

`UAttachments.h`

`UAttachments.cp`

See also

LAttachment

LCommander

LPane

LView

LWindow

LKeyScrollAttachment()

Purpose The constructor creates the object using the passed-in parameters.

Access Public

Prototype

```
LKeyScrollAttachment( LView *inViewToScroll );
LKeyScrollAttachment( LStream *inStream );
```

Parameters The parameters for these constructors are:

`LView*`	`inViewToScroll`	A pointer to a LView to set mViewToScroll to.
`LStream*`	`inStream`	A pointer to a stream object that contains the information to create the LBorderAttachment object.

ExecuteSelf()

Purpose This method handles keyboard navigation scrolling. If the key that is passed to this method via `ioParam` is not `Home`, `End`, `PageUp`, or `PageDown`, then mExecuteHost will be set to `true`, else it is set to `false`. If `true`, this indicates that the host should handle the key event. This is an override of ExecuteSelf() in LAttachment.

mViewToScroll

Purpose A pointer to a LView that we want to scroll using keyboard navigational keys.

Access Protected

Prototype

```
LView *mViewToScroll;
```

LLink

Overview

LLink is a PowerPlant class that is used for a simple linked list implementation.

Methods

The methods in this class are:

LLink() ~LLink()

GetLink() SetLink()

Data Members

The data members in this class are:

mLink

Operation

Use SetLink() and GetLink(). Like duh, that's all that's here. You typically don't use this class directly. It serves as a base class for custom subclasses. A typical subclass of LLink adds data members that store data you want to pass between threads in an LQueue object.

Source files

(Thread Classes)

`LLink.h`

`LLink.cp`

See also

LQueue

LLink()

Purpose

The constructor creates the object.

Access

Public

Prototype

```
LLink();
LLink (LLink* inLinkP);
```

Parameters

A pointer to a link.

~LLink()

Purpose The destructor destroys the `LLink` object.

Access Virtual

Prototype virtual `~LLink();`

GetLink()

Purpose Returns the contents of the link field.

Access Inline, Public

Prototype
```
inline LLink* GetLink() const;
```

Parameters None

Return The link

SetLink()

Purpose Changes the link field.

Access Inline, Public

Prototype
```
inline void SetLink(LLink* inLinkP);
```

Parameters A pointer to the link to set.

Return None

mLink

Purpose The link

Access Protected

Prototype `LLink* mLink;`

LListBox

Overview

LListBox is a PowerPlant class that is used for wrapping the Mac OS List Manager functionality.

Methods

The methods in this class are:

LListBox()
~LListBox()
ActivateSelf()
BeTarget()
ClickSelf()
DeactivateSelf()
DoNavigationKey()
DoTypeSelection()
DontBeTarget()
DrawSelf()
FindCommandStatus()
FocusDraw()
GetDescriptor()
GetDoubleClickMessage()
GetFocusBox()
GetLastSelectedCell()
GetMacListH()
GetValue()
HandleKeyPress()
HideSelf()
InitListBox()
MakeCellVisible()
MoveBy()
ObeyCommand()
ResizeFrameBy()
RestorePlace()
SavePlace()
SelectAllCells()
SelectOneCell()
SetDescriptor()
SetDoubleClickMessage()
SetValue()
ShowSelf()
UnselectAllCells()

Data Members

The data members in this class are:

mMacListH
mDoubleClickMessage

mFocusBox mTextTraitsID

mHasGrow

Operation For detailed information on the workings of this class, refer to *The PowerPlant Book*.

Source files (Pane Classes)

`LListBox.h`

`LListBox.cp`

Ancestors LAttachable

LBroadcaster

LCommander

LPane

LListBox()

Purpose The constructor creates objects from the passed-in parameters.

Access Public

Prototype

```
LListBox();
LListBox( const LListBox &inOriginal );
LListBox( const SPaneInfo &inPaneInfo,
     Boolean inHasHorizScroll,
     Boolean inHasVertScroll,
     Boolean inHasGrow,
     Boolean inHasFocusBox,
     MessageT inDoubleClickMessage,
     Int16 inTextTraitsID,
     Int16 inLDEFid,
     LCommander *inSuper );
LListBox( LStream *inStream );
```

Parameters The parameters for these constructors are:

`const LListBox&`	`inOriginal`	The object to be copied.
`const SPaneInfo&`	`inPaneInfo`	The reference to the Superpane.
`Boolean`	`inHasHorizScroll`	Set if horizontal scrolling.
`Boolean`	`inHasVertScroll`	Set if vertical scrolling.
`Boolean`	`inHasGrow`	Set if the box can grow.
`Boolean`	`inHasFocusBox`	Set if it has a focus box.
`MessageT`	`inDoubleClickMessage`	The double-click message.
`Int16`	`inTextTraitsID`	The resource ID for the text traits.
`Int16`	`inLDEFid`	The ID for the LDEF List Definition resource.
`LCommander*`	`inSuper`	The pointer to the Supercommander.
`LStream*`	`inStream`	The pointer to the stream object to read from.

~LListBox()

Purpose The destructor destroys the object.

Access Virtual, Public

Prototype

```
virtual ~LListBox();
```

ActivateSelf()

Purpose Activate the ListBox. The Toolbox shows the selection and scroll bars. This is an override of ActivateSelf() in LPane.

BeTarget()

Purpose ListBox is becoming the Target. This is an override of BeTarget() in LCommander.

ClickSelf()

Purpose Respond to Click inside an ListBox. This is an override of ClickSelf() in LPane.

DeactivateSelf()

Purpose Deactivate ListBox. The Toolbox hides the selection and scroll bars. This is an override of DeactivateSelf() in LPane.

DoNavigationKey()

Purpose Implements keyboard navigation by supporting selection change using the arrow keys, page up, page down, home, and end.

Access Virtual, Protected

Prototype

```
virtual void DoNavigationKey(
     const EventRecord &inKeyEvent );
```

Parameters The parameter for this method is:

`const EventRecord&`	`inKeyEvent`	The event record for the key press.

Return None

DoTypeSelection()

Purpose Change selection to the item beginning with the input characters.

Access Virtual, Protected

Prototype

```
virtual void DoTypeSelection(
     const EventRecord& inKeyEvent );
```

Parameters The parameter for this method is:

`const EventRecord&`	`inKeyEvent`	The event record for the key press.

Return None

DontBeTarget()

Purpose ListBox is no longer the Target. This is an override of DontBeTarget() in LCommander.

DrawSelf()

Purpose Draw ListBox. This is an override of DrawSelf() in LPane.

FindCommandStatus()

Purpose Pass back the status of a Command. This is an override of FindCommandStatus() in LCommander.

FocusDraw()

Purpose Focus drawing. This is an override to use the default Pen state and to set the TextTraits used by the ListBox. This is an override of FocusDraw() in LPane.

GetDescriptor()

Purpose Return the descriptor of a ListBox, which is the text of the first selected cell. The descriptor is an empty string if there are no selected cells. This method assumes that the cell data is text. This is an override of GetDescriptor() in LPane.

GetDoubleClickMessage()

Purpose Retrieve the double-click message.

Access Public

Prototype
```
MessageT GetDoubleClickMessage() const;
```

Parameters None

Return Returns the message for the double-click.

GetFocusBox()

Purpose Retrieve the focus box from mFocusBox.

Access Inline, Public

Prototype
```
LFocusBox* GetFocusBox();
```

Parameters None

Return Return the mFocusBox.

GetLastSelectedCell()

Purpose Pass back the last selected Cell in a ListBox.

Access Virtual, Public

Prototype
```
virtual Boolean GetLastSelectedCell(Cell &outCell);
```

Parameters The parameter for this method is:

`Cell&`	`outCell`	The reference to the last selected Cell.

Return Returns false if no cells are selected.

GetMacListH()

Purpose Retrieve the Mac OS list handle.

Access Public

Prototype

```
ListHandle GetMacListH() const;
```

Parameters None

Return Returns the list handle.

GetValue()

Purpose Return the value of a ListBox.

The "value" of a ListBox is the row number of the first selected cell, with the first row being number 0 (the ListManager uses zero-based numbering). If no cells are selected, the value is -1. This "value" makes sense for a ListBox with one column, by far the most common case.

This is an override of GetValue() in LPane.

HandleKeyPress()

Purpose ListBox supports keyboard navigation and type selection. This is an override of HandleKeyPress() in LCommander.

HideSelf()

Purpose ListBox is becoming invisible. This is an override of HideSelf() in LPane.

InitListBox()

Purpose Private initializer

Access Private

Prototype

```
void InitListBox(
     Boolean inHasHorizScroll,
     Boolean inHasVertScroll,
     Boolean inHasGrow,
     Boolean inHasFocusBox,
     MessageT inDoubleClickMessage,
     Int16 inTextTraitsID,
     Int16 inLDEFid );
```

Parameters The parameters for these constructors are:

`Boolean`	`inHasHorizScroll`	Set if horizontal scrolling.
`Boolean`	`inHasVertScroll`	Set if vertical scrolling.
`Boolean`	`inHasGrow`	Set if the box can grow.
`Boolean`	`inHasFocusBox`	Set if it has a focus box.
`MessageT`	`inDoubleClickMessage`	The double-click message.
`Int16`	`inTextTraitsID`	The resource ID for the text traits.
`Int16`	`inLDEFid`	The ID for the LDEF List Definition resource.

Return None

MakeCellVisible()

Purpose Scroll the ListBox as little as possible to move the specified Cell into view.

Access Virtual, Public

Prototype

```
virtual void MakeCellVisible( Cell inCell );
```

Parameters The parameter for this method is:

`Cell&`	`inCell`	The reference to the Cell.

Return None

MoveBy()

Purpose Move the location of the Frame by the specified amounts. Both PowerPlant and the List Manager store a location for the ListBox, so we must make sure that both locations are in synch. This is an override of MoveBy() in LPane.

ObeyCommand()

Purpose Respond to Command message. This is an override of ObeyCommand() in LCommander.

ResizeFrameBy()

Purpose Change the Frame size by the specified amounts. Both PowerPlant and the List Manager store a size for the ListBox, so we must make sure that both sizes are in synch. This is an override of ResizeFrameBy() in LPane.

RestorePlace()

Purpose Read size and location information stored in a Stream by the SavePlace() function.

Access Public

Prototype

```
virtual void RestorePlace( LStream *inPlace );
```

Parameters The parameter for this method is:

`LStream*`	`inPlace`	Reference to the stream to read from.

Return None

SavePlace()

Purpose Write size and location information to a Stream for later retrieval by the RestorePlace() function.

Access Public

Prototype
```
virtual void SavePlace( LStream *outPlace );
```

Parameters The parameter for this method is:

`LStream*`	`outPlace`	The reference to the stream to write to.

Return None

SelectAllCells()

Purpose Select all the cells in a ListBox.

Access Virtual, Public

Prototype
```
virtual void SelectAllCells();
```

Parameters None

Return None

SelectOneCell()

Purpose Select the specified Cell and deselect all others.

Access Virtual, Public

Prototype
```
virtual void SelectOneCell( Cell inCell );
```

Parameters The parameter for this method is:

`Cell&`	`inCell`	The reference to the Cell.

Return None

SetDescriptor()

Purpose Set the descriptor of a ListBox, which is the text of the first selected cell. Nothing happens if there are no selected cells.

This function assumes that the cell data is text. This is an override of SetDescriptor() in LPane.

SetDoubleClickMessage()

Purpose Set the double click message.

Access Public

Prototype

```
void SetDoubleClickMessage( MessageT inMessage);
```

Parameters The parameter for this method is:

`MessageT`	`inMessage`	The message to set for double click.

Return None

SetValue()

Purpose Set the value of a ListBox.

The "value" of a ListBox is the row number of the first selected cell, with the first row being number 0 (the ListManager uses zero-based numbering). If no cells are selected, the value is -1.

This "value" makes sense for a ListBox with one column, by far the most common case.

This function selects the cell in row "inValue" and column 1, deselecting any previously selected cells. This is an override of SetValue() in LPane.

ShowSelf()

Purpose ListBox is becoming visible. This is an override of ShowSelf() in LPane.

UnselectAllCells()

Purpose Unselect all the cells in a ListBox.

Access Public

Prototype
```
virtual void UnselectAllCells();
```

Parameters None

Return None

mMacListH

Purpose The Mac OS list handle.

Access Protected

Prototype
```
ListHandle mMacListH;
```

mDoubleClickMessage

Purpose Storage for the double click message.

Access Protected

Prototype
```
MessageT mDoubleClickMessage;
```

mFocusBox

Purpose Storage for the focus box.

Access Protected

Prototype
```
LFocusBox *mFocusBox;
```

mTextTraitsID

Purpose The resource ID for the text traits.

Access Protected

Prototype
```
ResIDT mTextTraitsID;
```

mHasGrow

Purpose Indicates whether the box can grow.

Access Protected

Prototype
```
Boolean mHasGrow;
```

LListener

Overview LListener is a PowerPlant class that is used for enabling your objects to respond to messages broadcasted by LBroadcaster objects.

Methods The methods in this class are:

LListener()	~LListener()
AddBroadcaster()	IsListening()
ListenToMessage()	RemoveBroadcaster()
StartListening()	StopListening()

Data Members The data members in this class are:

mBroadcasters	mIsListening

Operation An object that inherits from LListener receives messages from Broadcaster objects. To use LListener with your objects, inherit LListener in your class definition, and provide an implementation for the ListenToMessage() pure virtual method.

Source files (Feature Classes)

`LListener.h`

`LListener.cp`

See also LBroadcaster

LListener()

Purpose The default constructor enables listening for the object by setting mIsListening to true. The copy constructor copies the state of mIsListening, but does not copy any links to Broadcaster objects.

Access Public

Prototype

```
LListener();
```

```
LListener(const LListener &inOriginal);
```

Parameters None

Return None

~LListener()

Purpose The destructor removes this Listener from all the Broadcaster objects that it registered with.

Access Public, Virtual

Prototype
```
virtual ~LListener();
```

Parameters None

Return None

AddBroadcaster()

Purpose This method adds a link from a Broadcaster to a Listener. LBroadcaster::AddBroadcaster() will call this method.

```
theBroadcaster->AddListener(theListener); //correct
theListener->AddBroadcaster(theBroadcaster); //incorrect!
```

Access Protected

Prototype
```
void LListener::AddBroadcaster(
      LBroadcaster *inBroadcaster);
```

Parameters

`LBroadcaster*`	`inBroadcaster`	a pointer to an LBroadcaster object

Return None

Remarks Generally, you should not call this function directly.

IsListening()

Purpose This method is an accessor for the mIsListening data member.

Access Public

Prototype

```
Boolean IsListening() const;
```

Parameters None

Return A Boolean value of true indicates thatthe Listener object is currently listening.

ListenToMessage()

Purpose This method provides the message processing behavior for an LListener-inherited object. Your object inherits from LListener, and must provide some behavior for handling broadcasted messages. This method is where that behavior should be implemented.

Access Public, Pure virtual

Prototype

```
virtual void ListenToMessage(
      MessageT inMessage,
      void *ioParam) = 0;
```

Parameters

`MessageT`	`inMessage`	This parameter is the message that the Broadcaster is sending to the Listener.
`void*`	`ioParam`	This pointer is used to pass a parameter to the Listener message-processing code

Return None

Remarks Your object that inherits from LListener must implement this method, since it is a pure virtual method.

RemoveBroadcaster()

Purpose This method removes a Broadcaster from the list maintained by the LListener object.

```
theBroadcaster->RemoveListener(theListener); // correct
theListener->RemoveBroadcaster(theBroadcaster); // incorrect!
```

Access Protected

Prototype

```
void LListener::RemoveBroadcaster(
      LBroadcaster *inBroadcaster)
```

Parameters

`LBroadcaster*`	`inBroadcaster`	a pointer to the LBroadcaster object that is removed from the list

Return None

Remarks Normally, you should not call this method directly. LBroadcaster::RemoveListener() will call it.

StartListening()

Purpose This method causes the object to start listening to Broadcaster messages. This method is an accessor method for mIsListening.

Access Public

Prototype

```
void StartListening();
```

Parameters None

Return None

Remarks This method sets the mIsListening data member to true.

StopListening()

Purpose This method causes the object to stop listening to Broadcaster messages. This method is an accessor method for mIsListening.

Access Public

Prototype
```
void StopListening();
```

Parameters None

Return None

Remarks This method sets the mIsListening data member to false.

mBroadcasters

Purpose This data member is an array of pointers to Broadcaster objects.

Access Protected

Prototype
```
TArray<LBroadcaster*> mBroadcasters;
```

mIsListening

Purpose This data member tells whether the Listener is listening to Broadcasted messages.

Access Protected

Prototype
```
Boolean mIsListening;
```

LLongComparator

Overview LLongComparator is a PowerPlant class that is used for comparing `long` quantities.

Methods The methods in this class are:

LLongComparator()	~LLongComparator()
Clone()	Compare()
GetComparator()	IsEqualTo()

Data Members The data members in this class are:

sLongComparator

Operation For information on using this class, refer to LComparator.

Source files (Array Classes)

`LComparator.h`

`LComparator.cp`

Ancestors LComparator

LLongComparator()

Purpose The constructor is a placeholder, it doesn't do anything for this class.

Access Public

Prototype

```
LLongComparator();
```

Parameters None

~LLongComparator()

Purpose The destructor is a placeholder, it doesn't do anything for this class.

Access Virtual, Public

Prototype

```
~LLongComparator();
```

Clone()

Purpose This method behaves identically to Clone() in the base class LComparator.

Compare()

Purpose This method compares the contents of the two items to determine the comparison value.

Access Virtual, Public

Prototype

```
virtual Int32 Compare(
     const void* inItemOne,
     const void* inItemTwo,
     Uint32 inSizeOne,
     Uint32 inSizeTwo) const;
```

Parameters This method has the following parameters:

`const void*`	`inItemOne`	A data pointer to the first item.
`const void*`	`inItemTwo`	A data pointer to the second item.
`Uint32`	`inSizeOne`	Unused
`Uint32`	`inSizeTwo`	Unused

Return `Int32` that indicates one of the following:

- < 0 — object/data item 1 is less than object/data item 2

- 0 — object/data item 1 is equal to object/data item 2
- > 0 — object/data item 1 is greater than object/data item 2

GetComparator()

Purpose This method behaves identically to GetComparator() in the base class LComparator.

IsEqualTo()

Purpose This method detects whether two objects are equal.

Access Virtual, Public

Prototype

```
virtual Boolean IsEqualTo(
      const void* inItemOne,
      const void* inItemTwo,
      Uint32 inSizeOne,
      Uint32 inSizeTwo) const;
```

Parameters This method has the following parameters:

`const void*`	`inItemOne`	A data pointer to the first item.
`const void*`	`inItemTwo`	A data pointer to the second item.
`Uint32`	`inSizeOne`	Unused
`Uint32`	`inSizeTwo`	Unused

Return `Boolean` indicating `true` if the items are equal, `false` otherwise.

sLongComparator

Purpose This data member stores the pointer to the comparison "standard" object.

Access Protected

Prototype `static LLongComparator* sLongComparator;`

LMacTCPDNSOperation

Overview

LMacTCPDNSOperation is a PowerPlant class that is used for managing thread blocking for MacTCP DNS interface calls.

Methods

The methods in this class are:

LMacTCPDNSOperation() ~LMacTCPDNSOperation()

GetResultProc() Int_DNSCompletionProc()

Int_DNSCompletionProc2() WaitForCompletion()

Data Members

The data members in this class are:

mOperationListMem sMacTCPDNSCompletionProc

sMacTCPDNSPendingOperations; sDNSOperationDeleteQueue

Source files

(Networking Classes)

`UMacTCPSupport.h`

`UMacTCPSupport.cp`

See also

LGlobalsContext

LInterruptSafeListMember

StAsyncOperation

LMacTCPDNSOperation()

Purpose

The constructor creates the object.

Access

Public

Prototype `LMacTCPDNSOperation();`

Parameters None

~LMacTCPDNSOperation()

Purpose The destructor destroys the object.

Access Public

Prototype `~LMacTCPDNSOperation();`

GetResultProc()

Purpose Return the value of sMacTCPDNSCompletionProc.

Access Inline, Public

Prototype
```
inline ResultUPP GetResultProc();
```

Parameters None

Return The value of sMacTCPDNSCompletionProc.

Int_DNSCompletionProc()

Purpose The completion routine.

Access Static, Protected

Prototype
```
pascal void Int_DNSCompletionProc(
     hostInfo* inHostInfoPtr,
     Ptr inUserDataPtr
   );
```

Parameters This method has the following parameters:

`hostInfo*`	`inHostInfoPtr`	The pointer to the host info.
`Ptr`	`inUserDataPtr`	The user data pointer.

Return None

Int_DNSCompletionProc2()

Purpose The DNS completion procedure.

Access Static, Protected

Prototype

```
pascal void Int_DNSCompletionProc2(
     returnRec*inReturnRecPtr,
     Ptr inUserDataPtr );
```

Parameters This method has the following parameters:

`returnRec*`	`inReturnRecPtr`	The pointer to the return record.
`Ptr`	`inUserDataPtr`	The user data pointer.

Return None

WaitForCompletion()

Purpose Wait for the completion routine to execute.

Access Public

Prototype

```
void WaitForCompletion();
```

Parameters None

Return None

mOperationListMem

Purpose The operation list.

Access Private

Prototype
```
LOperationListMember * mOperationListMem;
```

sMacTCPDNSCompletionProc

Purpose The DNS completion routine pointer.

Access Static, Private

Prototype
```
static ResultUPP sMacTCPDNSCompletionProc;
```

sMacTCPDNSPendingOperations;

Purpose The MacTCP DNS pending operation.

Access Static, Private

Prototype
```
static Uint16 sMacTCPDNSPendingOperations;
```

sDNSOperationDeleteQueue

Purpose The DNS operation deletion queue.

Access Static, Private

Prototype
```
LDNSOperationDeleteQueue * sDNSOperationDeleteQueue;
```

LMacTCPTCPSendQueue

Overview `LMacTCPTCPSendQueue` is a PowerPlant class that is used for queuing data for asynchronous style data sending for MacTCP.

Methods The methods in this class are:

LMacTCPTCPSendQueue() ~LMacTCPTCPSendQueue()

Int_InternalSend() NotifyRelease()

Data Members The data members in this class are:

mWDS mMacTCPEndpoint

mOperation

Operation This class is used primarily for Open Transport-style "acked" data transfers which are not native to MacTCP.

Source files (Networking Classes)

`LMacTCPTCPSendQueue.h`

`LMacTCPTCPSendQueue.cp`

See also LSendQueue

LThread

LMacTCPTCPSendQueue()

Purpose The constructor creates the object.

Access Public

Prototype

```
LMacTCPTCPSendQueue(
     LMacTCPTCPEndpoint *inEndpoint );
```

Parameters This constructor has the following parameter:

`LMacTCPTCPEndpoint*`	`inEndpoint`	This is the endpoint.

~LMacTCPTCPSendQueue()

Purpose The destructor destroys the object.

Access Public

Prototype

```
virtual ~LMacTCPTCPSendQueue();
```

Int_InternalSend()

Purpose Sends are chained via the completion routine.

NOTE: This routine may be called at interrupt time.

Access Virtual, Protected

Prototype

```
virtual void Int_InternalSend();
```

Parameters None

Return None

NotifyRelease()

Purpose Returns the size of the pending data in the queue.

Access Virtual

Prototype

```
virtual void NotifyRelease(LSendData* inData);
```

Parameters This method has the following parameter:

`LSendData*`	`inData`	The data.

Return None

mWDS

Purpose Data storage.

Access Protected

Prototype
```
wdsEntry mWDS[5];
```

mMacTCPEndpoint

Purpose The MacTCP endpoint.

Access Protected

Prototype
```
LMacTCPTCPEndpoint* mMacTCPEndpoint;
```

mOperation

Purpose The operation.

Access Protected

Prototype
```
StMacTCPTCPSendOperation * mOperation;
```

LMacTCPUDPEndpoint

Overview

LMacTCPUDPEndpoint is a PowerPlant class that is an implementation of LTCPEndpoint that communicates with MacTCP.

Methods

The methods in this class are:

LMacTCPUDPEndpoint()	~LMacTCPUDPEndpoint()
AbortThreadOperation()	AckSends()
Bind()	DontAckSends()
GetLocalAddress()	GetState()
Int_HandleAsyncEvent()	Int_UDPNotifyProc()
IsAckingSends()	ReceiveFrom()
SendPacketData()	Unbind()

Data Members

The data members in this class are:

mUDPStream	mLocalAddress
mAckSends	mEndpointState
mReceiveBuffer	mReceiveBufferSize
mMessageQueue	mSharedPool
mSendQueue	sUDPNotifyUPP

Operation

We recommend that you do not make a subclass of this class.

Source files

(Networking Classes)

`LMacTCPUDPEndpoint.h`

`LMacTCPUDPEndpoint.cp`

See also

LTCPEndpoint

LUDPEndpoint

LGlobalsContext

LCleanupTask

LMacTCPUDPEndpoint()

Purpose The constructor creates the object.

Access Public

Prototype
```
LMacTCPUDPEndpoint(Uint32 inReceiveBufferSize);
```

Parameters This constructor has the following parameter:

`Uint32`	`inReceiveBufferSize`	This is the size of the receive buffer.

~LMacTCPUDPEndpoint()

Purpose The destructor destroys the object.

Access Virtual, Public

Prototype
```
virtual ~LMacTCPUDPEndpoint();
```

Parameters None

AbortThreadOperation()

Purpose Abort the thread operation.

Access Virtual, Public

Prototype
```
virtual void AbortThreadOperation(
     LThread * inThread);
```

Parameters This method has the following parameter:

`LThread*`	`inThread`	This is the thread.

Return None

AckSends()

Purpose Enable the sending of acknowledgements.

Access Virtual, Public

Prototype

```
virtual void AckSends();
```

Parameters None

Return None

Bind()

Purpose Bind an address.

Access Virtual, Public

Prototype

```
virtual void Bind(
     LInternetAddress& inLocalAddress,
     Uint32 inListenQueueSize,
     Boolean inReusePort );
```

Parameters This method has the following parameter:

`LInternetAddress&`	`inLocalAddress`	The local address.
`Uint32`	`inListenQueueSize`	The size of the listen queue.
`Boolean`	`inReusePort`	Whether to reuse the port.

Return None

DontAckSends()

Purpose Turn off the generation of acknowledgments.

Access Virtual, Public

Prototype
```
virtual void DontAckSends();
```

Parameters None

Return None

GetLocalAddress()

Purpose Obtain the local address, held in mLocalAddress.

Access Virtual, Public

Prototype
```
LInternetAddress* GetLocalAddress();
```

Parameters None

Return The local address.

GetState()

Purpose Retrieve the endpoint state, stored in mEndpointState.

Access Virtual, Public

Prototype
```
virtual EEndpointState GetState();
```

Parameters None

Return The value of mEndpointState.

Int_HandleAsyncEvent()

Purpose Called by NotifyProc to handle notifications for this endpoint. You will be notified via a Broadcast at primary task time of the event if necessary.

Warning: This routine will probably be called at interrupt time.

Access Protected

Prototype

```
void Int_HandleAsyncEvent(
     Uint16 inEventCode,
     ICMPReport*inEventCode,
     StreamPtr inStreamPtr );
```

Parameters This method has the following parameters:

`Uint16`	`inEventCode`	The local address.
`ICMPReport*`	`inEventCode`	The size of the listen queue.
`StreamPtr`	`inStreamPtr`	Whether to reuse the port.

Return None

Int_UDPNotifyProc()

Purpose

Access Static, Protected

Prototype

```
pascal static void Int_UDPNotifyProc(
     StreamPtr inStreamPtr,
     Uint16 inEventCode,
     Ptr inUserPtr,
     ICMPReport* inIcmpMsg );
```

Parameters This method has the following parameters:

`StreamPtr`	`inStreamPtr`	The stream pointer.
`ICMPReport*`	`inEventCode`	The size of the listen queue.
`Uint16`	`inUserPtr`	The pointer to the user area.
`ICMPReport*`	`inIcmpMsg`	The ICMP msg info.

Return None

IsAckingSends()

Purpose Determine whether sends are being acknowledged.

Access Virtual, Public

Prototype

```
virtual Boolean IsAckingSends();
```

Parameters None

Return Returns the value of mAckSends.

ReceiveFrom()

Purpose

Access Virtual, Public

Prototype

```
virtual void ReceiveFrom(
     LInternetAddress& outRemoteAddress,
     void* outDataBuffer,
     Uint32& ioDataSize,
     Uint32 inTimeoutSeconds );
```

Parameters This method has the following parameters:

`LInternetAddress&`	`outRemoteAddress`	The remote address.
`void*`	`outDataBuffer`	The data output buffer.
`Uint32&`	`ioDataSize`	The buffer size.
`Uint32`	`inTimeoutSeconds`	The timeout period to wait for data.

Return None

SendPacketData()

Purpose Send data.

Access Virtual, Public

Prototype

```
virtual void SendPacketData(
     LInternetAddress& inRemoteHost,
     void* inData,
     Uint32 inDataSize );
```

Parameters This method has the following parameters:

`LInternetAddress&`	`inRemoteHost`	The remote address.
`void*`	`inData`	The data input.
`Uint32`	`inDataSize`	The buffer size.

Return None

Unbind()

Purpose Unbind from a port.

Access Virtual, Public

Prototype `virtual void Unbind();`

Parameters None

Return None

mUDPStream

Purpose The UDP stream.

Access Protected

Prototype `StreamPtr mUDPStream;`

mLocalAddress

Purpose The local address.

Access Protected

Prototype `LInternetAddress * mLocalAddress;`

mAckSends

Purpose Whether to acknowledge sends or not.

Access Protected

Prototype `Boolean mAckSends;`

mEndpointState

Purpose The endpoint status.

Access Protected

Prototype `Uint16 mEndpointState;`

mReceiveBuffer

Purpose The receive buffer pointer.

Access Protected

Prototype
```
Ptr mReceiveBuffer;
```

mReceiveBufferSize

Purpose The receive buffer size in bytes.

Access Protected

Prototype
```
Uint32 mReceiveBufferSize;
```

mMessageQueue

Purpose The message queue.

Access Protected

Prototype
```
LNetMessageQueue * mMessageQueue;
```

mSharedPool

Purpose The shared memory pool.

Access Protected

Prototype
```
LSharedMemoryPool * mSharedPool;
```

mSendQueue

Purpose The send queue.

Access Protected

Prototype `LMacTCPUDPSendQueue * mSendQueue;`

sUDPNotifyUPP

Purpose The procedure pointer for the UDP notification procedure.

Access Static, Private

Prototype `UDPNotifyUPP sUDPNotifyUPP;`

LMacTCPUDPSendQueue

Overview LMacTCPUDPSendQueue is a PowerPlant class that is used for simplifying data sends.

Methods The methods in this class are:

LMacTCPUDPSendQueue()	~LMacTCPUDPSendQueue()
Int_InternalSend()	NotifyRelease()

Data Members The data members in this class are:

mWDS	mMacTCPUDPEndpoint
mOperation	

Source files (Networking Classes)

`LMacTCPUDPSendQueue.h`

`LMacTCPUDPSendQueue.cp`

See also LInterruptSafeListMember

LSendQueue

LMacTCPUDPSendQueue()

Purpose The constructor creates the object. It inherits from the LSendQueue() constructor.

Access Public

Prototype

```
LMacTCPUDPSendQueue(
     LMacTCPUDPEndpoint * inEndpoint);
```

Parameters This constructor has the following parameter:

`LMacTCPUDPEndpoint*`	`inEndpoint`	This is the endpoint.

~LMacTCPUDPSendQueue()

Purpose The destructor destroys the object.

Access Virtual, Public

Prototype

```
virtual ~LMacTCPUDPSendQueue()
```

Int_InternalSend()

Purpose Sends are chained via the completion routine.

NOTE: This routine may be called at interrupt time.

Access Virtual, Protected

Prototype

```
virtual void Int_InternalSend();
```

Parameters None

Return None

NotifyRelease()

Purpose Returns the size of the pending data in the queue.

Access Virtual, Protected

Prototype

```
void NotifyRelease(LSendData* inData);
```

Parameters This method has the following parameter:

`LSendData*`	`inData`	This is the data object.

Return None

mWDS

Purpose Data storage.

Access Protected

Prototype
```
wdsEntry mWDS[2];
```

mMacTCPUDPEndpoint

Purpose The MacTCP endpoint.

Access Protected

Prototype
```
LMacTCPUDPEndpoint * mMacTCPUDPEndpoint;
```

mOperation

Purpose The operation.

Access Protected

Prototype
```
StMacTCPUDPSendOperation * mOperation;
```

LMenu

Description

LMenu is a PowerPlant class that is used for managing a Mac OS menu. This class maintains a mapping between Menu items and Command numbers.

Methods

The methods in this class are:

LMenu()	~LMenu()
CommandFromIndex()	FindNextCommand()
GetMacMenuH()	GetMenuID()
GetNextMenu()	IndexFromCommand()
InsertCommand()	IsInstalled()
IsUsed()	ItemIsEnabled()
ReadCommandNumbers()	RemoveCommand()
RemoveItem()	SetCommand()
SetInstalled()	SetNextMenu()
SetUsed()	SyntheticCommandFromIndex()

Data Members

The data members in this class are:

mNextMenu	mMacMenuH
mMENUid	mNumCommands
mCommandNums	mIsInstalled
mUsed	

Operation

To learn more about effectively using the LMenu class, refer to *The PowerPlant Book* for a lengthy discussion on the topic.

Source Files

(Support Classes)

`LMenu.h`

`LMenu.cp`

LMenu()

Purpose The constructors create objects from the passed-in parameters.

Access Public

Prototypes

```
LMenu();
LMenu( ResIDT inMENUid);
LMenu( Int16 inMENUid,Str255 inTitle );
```

Parameters The parameters for these constructors are:

`ResIDT`	`inMENUid`	The resource ID for the menu.
`Int16`	`inMENUid`	The resource ID for the menu.
`Str255`	`inTitle`	The string to use for a title.

~LMenu()

Purpose The destructor destroys the LMenu object.

Access Public

Prototype

```
~LMenu();
```

CommandFromIndex()

Purpose Return the command number for a particular Menu item.

Access Public

Prototype

```
CommandT CommandFromIndex( Int16 inIndex ) const;
```

Parameters The parameter for this method is:

`Int16`	`inIndex`	The index to use to retrieve the command number.

Return The command number.

FindNextCommand()

Purpose Retrieve the next command.

Access Public

Prototype

```
Boolean FindNextCommand( Int16 &ioIndex,
     CommandT &outCommand ) const;
```

Parameters The parameters for this method are:

`Int16&`	`ioIndex`	The index to use to retrieve the command.
`CommandT&`	`outCommand`	The command.

Return Returns `true` if the command is found.

GetMacMenuH()

Purpose Retrieve OS menu handle.

Access Inline, Public

Prototype

```
MenuHandle GetMacMenuH() const;
```

Parameters None

Return A handle to the menu.

GetMenuID()

Purpose Retrieve the menu ID.

Access Inline, Public

Prototype
```
ResIDT GetMenuID() const;
```

Parameters None

Return The resource ID of the menu.

GetNextMenu()

Purpose Retrieve the next menu.

Access Inline, Protected

Prototype
```
LMenu* GetNextMenu() const;
```

Parameters None

Return A pointer to the LMenu object.

IndexFromCommand()

Purpose Return the Menu item index number for a particular command number Return 0 if the command number is not used for this Men

Access Public

Prototype
```
Int16 IndexFromCommand(
     CommandT inCommand ) const;
```

Parameters The parameter for this method is:

`CommandT&`	`inCommand`	The command.

Return `Int16` containing the index.

InsertCommand()

Purpose Insert an item with the specified text and command number after-particular item.

Access Public

Prototype

```
void InsertCommand( ConstStringPtr inItemText,
     CommandT inCommand,
     Int16 inAfterItem );
```

Parameters The parameters for this method are:

`ConstStringPtr`	`inItemText`	The index to use to retrieve the command.
`CommandT&`	`inCommand`	The command.
`Int16`	`inAfterItem`	The item to insert after.

Return None

Remarks This function does not support adding multiple menu items using "Return" or "Semicolon" characters within `inItemText`.

IsInstalled()

Purpose Retrieves the value of mIsInstalled.

Access Inline, Public

Prototype

```
Boolean IsInstalled() const;
```

Parameters None

Return The value of mIsInstalled.

IsUsed()

Purpose Retrieves the value of mUsed.

Access Inline, Public

Prototype
```
Boolean IsUsed() const;
```

Parameters None

Return The value of mUsed.

ItemIsEnabled()

Purpose Return whether an item at the specified position is enabled.

Access Public

Prototype
```
Boolean ItemIsEnabled( Int16 inIndex ) const;
```

Parameters The parameter for this method is:

`Int16`	`inIndex`	The index to check.

Return Returns `true` if enabled, else `false`.

ReadCommandNumbers()

Purpose Get command numbers from the 'Mcmd' resource. A 'Mcmd'' resource with the same ID as the 'MENU' resource contains a list of command numbers. The format of a 'Mcmd' resource is:

- Number of Commands: n(2 bytes)
- Command Num for Item 1(4 bytes)
- Command Num for Item n(4 bytes)

Access Protected

Prototype `void ReadCommandNumbers();`

Parameters None

Return None

RemoveCommand()

Purpose Remove the Menu item with the specified command.

Access Public

Prototype `void RemoveCommand( CommandT inCommand );`

Parameters The parameter for this method is:

`CommandT`	`inCommand`	The command for the menu item to remove.

Return None

RemoveItem()

Purpose Remove the Menu item at the specified position.

Access Public

Prototype `void RemoveItem( Int16 inItemToRemove );`

Parameters The parameter for this method is:

`Int16`	`inItemToRemove`	The menu item to remove.

Return None

SetCommand()

Purpose Set the command number for a Menu item.

Access Public

Prototype

```
void SetCommand(
     Int16 inIndex,
     CommandT inCommand );
```

Parameters The parameters for this method are:

`Int16`	`inIndex`	The index to use to set the command.
`CommandT&`	`inCommand`	The command.

Return None

SetInstalled()

Purpose Sets the value of the mIsInstalled data member.

Access Public

Prototype

```
void SetInstalled(Boolean inInstalled);
```

Parameters The parameter for this method is:

`Boolean`	`inInstalled`	The value to set.

Return None

SetNextMenu()

Purpose Set the next Menu.

Access Inline, Protected

Prototype `void SetNextMenu( LMenu *inMenu );`

Parameters The parameter for this method is:

`LMenu*`	`inMenu`	The pointer to the Menu to set.

Return None

SetUsed()

Purpose This method sets the value of the mUsed data member.

Access Public

Prototype `void SetUsed( Boolean inUsed );`

Parameters The parameter for this method is:

`Boolean`	`inUsed`	The value to set the data member to.

Return None

SyntheticCommandFromIndex()

Purpose Retrieve the synthetic command number for a particular Menu item. A synthetic command number has the MENU id in the high 16 bits and the item number in the low 16 bits, and the resulting 32-bit number is negated (to distinguish it from regular command numbers). The synthetic command number is the negative of the value that would be returned by the Toolbox trap `MenuSelect()` for the menu item.

Access Public

Prototype
```
CommandT SyntheticCommandFromIndex(
     Int16 inIndex ) const;
```

Parameters The parameter for this method is:

`Int16`	`inIndex`	The index for the Menu item.

Return The command for the Menu item.

mNextMenu

Purpose Storage for the next menu.

Access Protected

Prototype
```
LMenu *mNextMenu;
```

mMacMenuH

Purpose The Mac OS menu handle.

Access Protected

Prototype
```
MenuHandle mMacMenuH;
```

mMENUid

Purpose The menu ID.

Access Protected

Prototype
```
ResIDT mMENUid;
```

mNumCommands

Purpose The number of commands.

Access Protected

Prototype Int16 mNumCommands;

mCommandNums

Purpose The commadn numbers.

Access Protected

Prototype
```
CommandT **mCommandNums;
```

mIsInstalled

Purpose Storage for detecting installation.

Access Protected

Prototype
```
Boolean mIsInstalled;
```

mUsed

Purpose Indicates whether used or not.

Access Protected

Prototype
```
Boolean mUsed;
```

LMenuBar

Description LMenuBar is a PowerPlant class that is used for managing the menu bar.

Methods The methods in this class are:

LMenuBar()	~LMenuBar()
CouldBeKeyCommand()	FetchMenu()
FindCommand()	FindKeyCommand()
FindMenuItem()	FindNextCommand()
FindNextMenu()	GetCurrentMenuBar()
InstallMenu()	MenuCommandSelection()
RemoveMenu()	

Data Members The data members in this class are:

sMenuBar	mMenuListHead

Operation For detailed information on how to use this class, refer to *The PowerPlant Book*.

Source files (Support Classes)

`LMenuBar.h`

`LMenuBar.cp`

LMenuBar()

Purpose The constructor creates the menu bar.

Access Public

Prototype

```
LMenuBar( ResIDT inMBARid );
```

Parameters This constructor has the following parameter:

`ResIDT`	`inMBARid`	The resource ID of the menu bar.

~LMenuBar()

Purpose The destructor destroys the object.

Access Virtual, Public

Prototype
```
virtual ~LMenuBar();
```

CouldBeKeyCommand()

Purpose Return whether the keystoke could be a key equivalent for a menu command. Overrid this to implement keyboard equivalents that use modifier keys other than just the Command key.

Access Virtual, Public

Prototype
```
virtual Boolean CouldBeKeyCommand(
     const EventRecord &inKeyEvent ) const;
```

Parameters This method has the following parameter:

`const EventRecord&`	`inKeyEvent`	The event for the key down.

Return This function returns true if the command key is down.

FetchMenu()

Purpose Return the Menu object for the specified MENU resource ID Returns nil if there is no such Menu object

Access Public

Prototype
```
LMenu *FetchMenu( ResIDT inMENUid ) const;
```

Parameters This method has the following parameter:

`ResIDT`	`inMENUid`	The menu resource ID.

Return A pointer to an LMenu.

FindCommand()

Purpose Retrieve the Command number corresponding to a Menu (ID, item) pair.

Access Public

Prototype

```
CommandT FindCommand( ResIDT inMENUid,
     Int16 inItem ) const;
```

Parameters This method has the following parameters:

`ResIDT`	`inMENUid`	The menu resource ID.
`Int16`	`inItem`	The menu item.

Return Returns the command number.

FindKeyCommand()

Purpose Return the Command number corresponding to a keystroke. Call this function when CouldBeKeyCommand() is true. You should override this to implement keyboard equivalents that use modifier keys other than just the Command key.

Access Virtual, Public

Prototype

```
virtual CommandT FindKeyCommand(
     const EventRecord &inKeyEvent,
     Int32 &outMenuChoice ) const;
```

Parameters

`const EventRecord &`	`inKeyEvent`	The key down event.
`Int32&`	`outMenuChoice`	The menu choice.

Return The command number. Returns `cmd_Nothing` if the keystroke is not a menu equivalent

Remarks This function calls the Toolbox routine MenuKey to find the associated menu item, if any. Override this function (as well as CouldBeKeyCommand) to implement key equivalents that use other modifier keys, such as Option, Shift, or Control.

FindMenuItem()

Purpose Passes back the MENU id, MenuHandle, and item number corresponding to a Command number.

If the Command is not associated with any item in the MenuBar, `outMENUid` is 0, `outMenuHandle` is `nil`, and `outItem` is 0.

Access Public

Prototype

```
void FindMenuItem(
     CommandT inCommand,
     ResIDT &outMENUid,
     MenuHandle &outMenuHandle,
     Int16 &outItem);
```

Parameters This method has the following parameter:

`ResIDT`	`outMENUid`	The menu resource ID.
`Int16`	`outItem`	The menu item.

CommandT	inCommand	The command.
MenuHandle	outMenuHandle	A handle to the menu.

Return None

Remarks None

FindNextCommand()

Purpose Find the next command.

Access Public

Prototype

```
Boolean FindNextCommand(
     Int16 &ioIndex,
     MenuHandle &ioMenuHandle,
     LMenu* &ioMenu,
     CommandT &outCommand );
```

Parameters This method has the following parameters:

`LMenu*`	`ioMenu`	The menu pointer.
`Int16&`	`ioIndex`	The index.
`CommandT&`	`outCommand`	The command.
`MenuHandle&`	`ioMenuHandle`	A handle to the menu.

Return Return `true` if the command is found, else return `false`.

FindNextMenu()

Purpose Find the next menu in the menu bar.

Access Public

Prototype `Boolean FindNextMenu( LMenu* &ioMenu ) const;`

Parameters This method has the following parameter:

`LMenu*&`	`ioMenu`	The menu pointer.

Return Return `true` if a menu was found, else return `false`.

GetCurrentMenuBar()

Purpose Retrieve the menu bar.

Access Static, Public

Prototype
```
static LMenuBar* GetCurrentMenuBar();
```

Parameters None

Return The pointer to the menu bar.

InstallMenu()

Purpose Install a Menu object in the MenuBar

Access Public

Prototype
```
void InstallMenu(
     LMenu *inMenuToInstall,
     ResIDT ioMenuHandle );
```

Parameters This method has the following parameters:

`LMenu*`	`inMenuToInstall`	The menu to install in the menu bar.
`ResIDT`	`ioMenuHandle`	The resource ID for the menu.

Return None

MenuCommandSelection()

Purpose Handle menu selection with the Mouse and return the command number for the item chosen. Override this method to implement alternative menu selection behavior.

Access Virtual, Public

Prototype

```
virtual CommandT MenuCommandSelection(
     const EventRecord &inMouseEvent,
     Int32 &outMenuChoice ) const;
```

Parameters This method has the following parameters:

`const EventRecord&`	`inMouseEvent`	The mouse event information.
`Int32&`	`outMenuChoice`	The menu choice.

Return The command number.

RemoveMenu()

Purpose Remove a Menu object from the menu bar.

Access Public

Prototype

```
void RemoveMenu( LMenu *inMenuToRemove );
```

Parameters This method has the following parameters:

`LMenu*`	`inMenuToRemove`	The menu to remove.

Return None

sMenuBar

Purpose Storage for the menu bar pointer.

Access Protected

Prototype `static LMenuBar *sMenuBar;`

mMenuListHead

Purpose The menu list head.

Access Protected

Prototype `LMenu *mMenuListHead;`

LMouseTracker

Overview LMouseTracker is a PowerPlant class that is used for periodically tracking the mouse.

Methods The methods in this class are:

LMouseTracker() ~LMouseTracker()

SpendTime()

Data Members There are no data members for this class.

Operation You can use this class to do things based on the location of the mouse pointer.

Source files (Support Classes)

`LMouseTracker.h`

`LMouseTracker.cp`

Ancestors LPeriodical

LMouseTracker()

Purpose The constructor is a placeholder (does nothing).

Access Public

Prototype

```
LMouseTracker();
```

~LMouseTracker()

Purpose The destructor is a placeholder (does nothing).

Access Virtual, Public

Prototype

```
virtual ~LMouseTracker();
```

Parameters None

SpendTime()

Purpose Track the mouse. This is an override of SpendTime() in LPeriodical.

LMovieController

Overview

LMovieController is a PowerPlant class that is used for creating, drawing, and disposing of a standard Mac OS QuickTime movie controller.

Methods

The methods in this class are:

LMovieController()	~LMovieController()
DrawSelf()	SpendTime()

Data Members

The data members in this class are:

mMovie	mMovieController

Operation

You would typically use this class in conjunction with the UQuickTime object if your application supports QuickTime movies. Since this class inherits from LPeriodical, it receives and can process every event retrieved by the event loop.

Source files

(Pane Classes)

`UQuickTime.h`

`UQuickTime.cp`

Ancestors

LPane

LPeriodical

LMovieController()

Purpose

The constructors create objects from the passed-in parameters.

Access

Public

Prototype

```
LMovieController();
LMovieController( const SPaneInfo &inPaneInfo,
     Movie inMovie );
```

Parameters This method has the following parameters:

`const SPaneInfo&`	`inPaneInfo`	The reference for the SuperView that the movie controller belongs to.
`Movie`	`inMovie`	The resource ID for the movie that will be displayed with the controller.

~LMovieController()

Purpose The destructor destroys the object.

Access Virtual, Public

Prototype
```
virtual ~LMovieController();
```

DrawSelf()

Purpose This method is an override of the base class method DrawSelf() in LPane. It draws the movie controller.

SpendTime()

Purpose This method is an override of the base class method DrawSelf() in LPeriodical. It handles events in the movie controller.

mMovie

Purpose This data member stores the resource ID of the movie for the controller.

Access Protected

Prototype
```
Movie mMovie;
```

mMovieController

Purpose This data member stores the `MovieController`.

Access Protected

Prototype

```
MovieController mMovieController;
```

LMutexSemaphore

Overview LMutexSemaphore is a PowerPlant class that is used for implementing a mutually-exclusive semaphore.

Methods The methods in this class are:

LMutexSemaphore()	~LMutexSemaphore()
Signal()	Wait()

Data Members The data members in this class are:

mOwner	mNestedWaits

Operation If a mutually-exclusive semaphore is raised, only one thread may access the flagged data. When the semaphore is lowered, if there are waiting threads, opnly one waiting thread is returned to the ready state.

Source files (Threads Classes)

`LMutexSemaphore.h`

`LMutexSemaphore.cp`

See also LSemaphore

LMutexSemaphore()

Purpose The constructor creates the object.

Access Public

Prototype

```
LLMutexSemaphore();
LMutexSemaphore(Boolean owned);
```

Parameters Indicate whether the semaphore is already owned or not. If owned is true, the semaphore is marked as belonging to the current thread.

~LMutexSemaphore()

Purpose The destructor destroys the LMutexSemaphore object.

Access Virtual

Prototype
```
virtual ~LMutexSemaphore();
```

Signal()

Purpose This is an override of Signal() in LSemaphore. If the owning semaphore called Wait() more than once, this function merely decrements a usage count. Else, if any threads are waiting on the semaphore, one of them is unblocked. If the current thread doesn't own the semaphore, the errSemaphoreNotOwner exception is thrown.

Wait()

Purpose This is an override of Wait() in LSemaphore.

mOwner

Purpose The owner.

Access Protected

Prototype
```
LThread*mOwner;
```

mNestedWaits

Purpose The number of nested waits

Access Protected

Prototype
```
Uint32 mNestedWaits;
```

LNetMessageQueue

Overview LNetMessageQueue is a PowerPlant class that is used for queuing of messages, typically generated at interrupt time, that will be broadcast to registered listeners at primary task time via a periodical.

Methods The methods in this class are:

LNetMessageQueue() ~LNetMessageQueue()

SpendTime()

Data Members The data members in this class are:

mBroadcaster

Source files (Networking Classes)

`LNetMessageQueue.h`

`LNetMessageQueue.cp`

See also LBroadcaster

LInterruptSafeList

LPeriodical

LNetMessageQueue()

Purpose The constructor creates the object.

Access Public

Prototype

```
LNetMessageQueue(LBroadcaster &inBroadcaster);
```

Parameters The parameter for this constructor is:

`LBroadcaster&`	`inBroadcaster`	The reference to the Broadcaster.

~LNetMessageQueue()

Purpose The destructor destroys the object.

Access Virtual, Public

Prototype

```
virtual ~LNetMessageQueue();
```

SpendTime()

Purpose This is an override of SpendTime() in LPeriodical.

mBroadcaster

Purpose The broadcaster.

Access Protected

Prototype

```
LBroadcaster * mBroadcaster;
```

LODBCApplication

Overview This PowerPlant class is designed to help with the essentials of buil ding anapplicationwith ODBC database support.

Methods The methods in this class are:

LODBCApplication() ~LODBCApplication()

OpenDatabase()

Data Members The data members in this class are:

mDatabase

Operation Your application should inherit from this class if you want to take advantage of the utilities provided by this class.

Source files (_Advanced Classes:Database Access)

`LODBCApplication.h`

`LODBCApplication.cp`

Ancestors LDBApplication

LODBCApplication()

Purpose The default constructor initializes the object.

Access Public

Prototype

```
LODBCApplication();
```

Parameters None

~LODBCApplication()

Purpose The destructor destroys the object.

Access Public, Virtual

Prototype
```
virtual ~LODBCApplication();
```

Parameters None

OpenDatabase()

Purpose Select the database, login and open the database.

Access Virtual, Public

Prototype
```
virtual void OpenDatabase();
```

Parameters None

Return None

mDatabase

Purpose This data member stores the database pointer.

Access Protected

Prototype
```
LODBCDatabase *mDatabase;
```

LOffscreenView

Overview LOffscreenView is a PowerPlant class that is used for offscreen drawing. It creates a temporary GWorld everytime you draw.

Methods The methods in this class are:

LOffscreenView()	~LOffscreenView()
Draw()	DrawOffscreen()
EstablishPort()	SubImageChanged()

Data Members The data members in this class are:

mOffscreenWorld	mDrawingSelf

Operation If you want a view or panes within the view to use a GWorld that lasts for more than one update, use LGWorld instead.

Source files (Pane Classes)

`LOffscreenView.h`

`LOffscreenView.cp`

Ancestors LAttachable

LPane

LView

LOffscreenView()

Purpose The constructors create objects from the passed-in parameters.

Access Public

Prototype

```
LOffscreenView();
```

```
LOffscreenView(
     const SPaneInfo &inPaneInfo,
     const SViewInfo &inViewInfo );
LOffscreenView( LStream *inStream );
```

Parameters The parameters for these constructors are:

`const SPaneInfo&`	`inPaneInfo`	The Superpane info.
`const SViewInfo&`	`inViewInfo`	The Superview info.
`LStream*`	`inStream`	The stream to read from.

~LOffscreenView()

Purpose The destructor destroys the object.

Access Virtual, Public

Prototype

```
virtual ~LOffscreenView();
```

Draw()

Purpose Draw contents of View offscreen and then copy them onscreen. First, we try to draw offscreen using temporary memory. If that fails, we try to draw offscreen using application memory. If all else fails, we call the inherited version to draw directly to the screen. This is an override of Draw() in LPane.

DrawOffscreen()

Purpose Draw contents of View to an Offscreen World, then copy bits all at once to the normal port.

Use the given flags when creating the offscreen GWorld.

Access Virtual, Protected

Prototype

```
virtual void DrawOffscreen(
     RgnHandle inSuperDrawRgnH,
     GWorldFlags inFlags );
```

Parameters This method has the following parameter:

`RgnHandle`	`inSuper-DrawRgnH`	The region handle.
`GWorldFlags`	`inFlags`	The flags for the GWorld creation.

Return None

EstablishPort()

Purpose Set current port to the OffscreenView.

If this View is being drawn, the current port should be the Offscreen GWorld. Otherwise, some subpane is trying to draw itself directly (not in response to an update event) so we call the inherited function to do the normal thing.

Access Virtual, Public

Prototype

```
virtual Boolean EstablishPort();
```

Parameters None

Return None

SubImageChanged()

Purpose Notification that the Image of some SubView changed size, location, or scroll units. In this override, we pass the notification up the chain. This is because the LOffscreenView isn't really the logical superview of its subpanes; it's just in the hierarchy to draw offscreen. So, we pass the notification up to the logical superview of its subpanes.

Access Virtual, Public

Prototype `virtual void SubImageChanged( LView* inSubView );`

Parameters This method has the following parameter:

`LView*`	`inSubView`	The pointer to the view.

Return None

mOffscreenWorld

Purpose The GWorld pointer.

Access Protected

Prototype `GWorldPtr mOffscreenWorld;`

mDrawingSelf

Purpose Storage for the information about self drawing.

Access Protected

Prototype `Boolean mDrawingSelf;`

LOperationListMember

Overview LOperationListMember is a PowerPlant class that simplifies operation list management.

Methods The methods in this class are:

LOperationListMember() ~LOperationListMember()

Data Members The data members in this class are:

mOperation

Source files (Networking Classes)

`LNetworking.h`

`LNetworking.cp`

See also LInterruptSafeListMember

StAsyncOperation

LOperationListMember()

Purpose The constructor creates the object.

Access Public

Prototype

```
LOperationListMember(
    StAsyncOperation *inOperation );
```

Parameters This constructor has the following parameter:

`StAsyncOperation*`	`inOperation`	The operation.

~LOperationListMember()

Purpose The destructor destroys the object.

Access Virtual, Public

Prototype
```
virtual ~LOperationListMember();
```

mOperation

Purpose The operation.

Access Static

Prototype
```
StAsyncOperation *mOperation;
```

LOverlappingView

Overview LOverlappingView is a PowerPlant class that allows you to create a container (superview) for another view. LOverlappingView is a View which draws properly when sibling views (views within the same superview) overlap its Frame.

Methods The methods in this class are:

LOverlappingView() ~LOverlappingView()

FocusDraw()

Data Members There are no data members in this class.

Operation Use LOverlappingView as a container (superview) for another view.

Source files (Pane Classes)

`LOverlappingView.h`

`LOverlappingView.cp`

Ancestors LAttachable

LPane

LView

LOverlappingView()

Purpose The constructor creates the object.

Access Public

Prototype

```
LOverlappingView();
LOverlappingView( LStream *inStream );
```

Parameters These constructors have the following parameter:

`LStream*`	`inStream`	The stream to read from.

~LOverlappingView()

Purpose The destructor destroys the object.

Access Virtual, Public

Prototype

```
virtual ~LOverlappingView();
```

FocusDraw()

Purpose Set up coordinates system and clipping region. An LOverlapping-View sets the clipping region to the revealed portion of its Frame minus the Frames of any sibling Views that are in front of it. This method is an override of FocusDraw() in LPane.

LPaintAttachment

Overview

LPaintAttachment is a PowerPlant class that is used for painting rectangles within the Frame of a Pane.

Methods

The methods in this class are:

LPaintAttachment()

ExecuteSelf()

Data Members

The data members in this class are:

mPenState

mForeColor

mBackColor

Operation

This objects allows you to paint a rectangle within the Frame of a Pane, for use only with the `msg_DrawOrPrint` message.

The rectangle is painted using the specified `PenState` settings and foreground and background `RGBColor` values. The painted rectangle is inset from the Frame by the size of the `pnSize` field of the `PenState`. This lets you use an LPaintAttachment in conjunction with a LBorderAttachment to draw a filled rectangle.

Source files

(Utility Classes)

`UAttachments.h`

`UAttachments.cp`

See also

LAttachment

LBorderAttachment

LPane

LPaintAttachment()

Purpose The constructors provide two ways to create LPaintAttachment objects. You may create a stream-based object or pass the itemized values for each required value.

Access Public

Prototype

```
LPaintAttachment( PenState *inPenState,
     RGBColor *inForeColor,
     RGBColor *inBackColor,
     Boolean inExecuteHost);
LPaintAttachment( LStream *inStream );
```

Parameters The parameters for these constructors are:

`PenState*`	`inPenState`	A pointer to a PenState.
`RGBColor*`	`inForeColor`	A pointer to an RGBColor.
`RGBColor*`	`inBackColor`	A pointer to an RGBColor.
`Boolean`	`inExecuteHost`	A Boolean indicating the value to be set for mExecuteHost.
`LStream*`	`inStream`	A pointer to a stream object that contains the information to create the LPaintAttachment object.

ExecuteSelf()

Purpose This method paints a rectangle. This is an override of ExecuteSelf() in LAttachment.

mPenState

Purpose This member holds the pen information.

Access Protected

Prototype `PenState mPenState;`

mForeColor

Purpose This member holds the foreground color information.

Access Protected

Prototype `RGBColor mForeColor;`

mBackColor

Purpose This member holds the background color information.

Access Protected

Prototype `RGBColor mBackColor;`

LPane

Overview

LPane is the fundamental visual class in PowerPlant. Every area you draw in, and everything you draw, descends from LPane. There are LPane derivative classes for text, pictures, lists, tables, controls and more.

Methods

The methods in this class are:

LPane()
~LPane()
Activate()
ActivateSelf()
AdaptToNewSurroundings()
AdaptToSuperFrameSize()
AdaptToSuperScroll()
AdjustCursor()
AdjustCursorSelf()
ApplyForeAndBackColors()
CalcLocalFrameRect()
CalcPortFrameRect()
Click()
ClickSelf()
ClickTimesAreClose()
Contains()
CountPanels()
Deactivate()
DeactivateSelf()
Disable()
DisableSelf()
DontRefresh()
Draw()
DrawSelf()
Enable()
EnableSelf()
EventMouseUp()
FindConstPaneByID()
FindDeepSubPaneContaining()
FindPaneByID()
FindShallowSubPaneContaining()
FindSubPaneHitBy()
FinishCreate()
FinishCreateSelf()
FocusDraw()
FocusExposed()
GetActiveState()
GetClickCount()
GetDefaultView()
GetDescriptor()

GetEnabledState()
GetFrameBinding()
GetFrameLocation()
GetFrameSize()
GetLastPaneClicked()
GetLastPaneMoused()
GetLocalUpdateRgn()
GetMacPort()
GetPaneID()
GetSuperView()
GetUserCon()
GetValue()
GetVisibleState()
GlobalToPortPoint()
Hide()
HideSelf()
InitPane()
InvalPortRect()
InvalPortRgn()
IsActive()
IsAreaInQDSpace()
IsEnabled()
IsHitBy()
IsVisible()
LocalToPortPoint()
MouseEnter()
MouseLeave()
MouseWithin()
MoveBy()
PlaceInSuperFrameAt()
PlaceInSuperImageAt()
PointIsInFrame()
PointsAreClose()
PortToGlobalPoint()
PortToLocalPoint()
PrintPanel()
PrintPanelSelf()
PutInside()
Refresh()
ResizeFrameBy()
ResizeFrameTo()
RestorePlace()
SavePlace()
ScrollToPanel()
SetDefaultView()
SetDescriptor()
SetForeAndBackColors()
SetFrameBinding()
SetLastPaneMoused()
SetPaneID()

SetRefreshAllWhenResized() SetUserCon()
SetValue() Show()
ShowSelf() SuperActivate()
SuperDeactivate() SuperDisable()
SuperEnable() SuperHide()
SuperPrintPanel() SuperShow()
UpdateClickCount() UpdatePort()
ValidPortRect() ValidPortRgn()

Data Members

The data members in this class are:

sDefaultView sLastPaneClicked
sLastPaneMoused sWhenLastMouseUp
sWhenLastMouseDown sWhereLastMouseDown
sClickCount mPaneID
mFrameSize mFrameLocation
mFrameBinding mUserCon
mSuperView mVisible
mActive mEnabled
mRefreshAllWhenResized

Operation

LPane is one of the central classes to the magic of PowerPlant. In order to learn how to use it effectively, you should read about it in *The PowerPlant Book* on your documentation CD.

Source files

(`Pane` Classes)

`LPane.h`

`LPane.cp`

See also LAttachable

LView

LPane()

Purpose The constructor creates the object using the passed-in parameters.

Access Public

Prototype

```
LPane();
LPane( const LPane &inOriginal );
LPane( const SPaneInfo &inPaneInfo );
LPane( LStream *inStream );
```

Parameters The parameters for these contructors are:

`const LPane&`	`inOriginal`	The object to copy.
`const SPaneInfo&`	`inPaneInfo`	The Superpane's info.
`LStream*`	`inStream`	The stream to read from.

~LPane()

Purpose The destructor destroys the object.

Access Virtual, Public

Prototype

```
virtual ~LPane();
```

Activate()

Purpose Activate a pane.

Access Virtual, Public

Prototype `virtual void Activate();`

Parameters None

Return None

ActivateSelf()

Purpose This is called when a pane is being activated. You should override this method for pane classes that change appearance when activated.

Access Virtual, Protected

Prototype `virtual void ActivateSelf();`

Parameters None

Return None

AdaptToNewSurroundings()

Purpose Adjust the state of the pane when installed in a new parent pane.

Access Virtual, Public

Prototype `virtual void AdaptToNewSurroundings();`

Parameters None

Return None

AdaptToSuperFrameSize()

Purpose Adjust the state of the pane when the size of the superview's frame changes by the specified amounts.

Access Virtual, Public

Prototype

```
virtual void AdaptToSuperFrameSize(
     Int32 inSurrWidthDelta,
     Int32 inSurrHeightDelta,
     Boolean inRefresh);
```

Parameters The parameters for this method are:

`Int32`	`inSurrWidthDelta`	The change in width.
`Int32`	`inSurrHeightDelta`	The change in height.
`Boolean`	`inRefresh`	Indicate whether to refresh or not.

Return None

AdaptToSuperScroll()

Purpose Adjust the state of the pane when its superview scrolls by the specified amounts.

Access Virtual, Public

Prototype

```
virtual void AdaptToSuperScroll(
     Int32 inHorizScroll,
     Int32 inVertScroll);
```

Parameters The parameters for this method are:

`Int32`	`inHorizScroll`	The width amount to scroll.
`Int32`	`inVertScroll`	The height amount to scroll.

Return None

AdjustCursor()

Purpose Wrapper function for setting the cursor shape.

Access Virtual, Public

Prototype

```
virtual void AdjustCursor(
     Point inPortPt,
     const EventRecord &inMacEvent );
```

Parameters The parameters for this method are:

`Point`	`inPortPt`	The point in Port coordinates.
`const EventRecord&`	`inMacEvent`	The event record.

Return None

AdjustCursorSelf()

Purpose Set the cursor shape when the cursor is inside a pane.

Access Virtual, Public

Prototype

```
virtual void AdjustCursorSelf(
     Point inPortPt,
     const EventRecord& inMacEvent);
```

Parameters The parameters for this method are:

`Point`	`inPortPt`	The point in Port coordinates.
`const EventRecord&`	`inMacEvent`	The event record. Use the `inMacEvent->modifiers` if the cursor shape depends on whether modifier keys (such as Option) are down.

Return None

ApplyForeAndBackColors()

Purpose Set the foreground and background colors of the current port. The pane must already be focused.

Access Virtual, Protected

Prototype
```
virtual void ApplyForeAndBackColors() const;
```

Parameters None

Return None

CalcLocalFrameRect()

Purpose Calculate the pane's frame rectangle in local (Image) coordinates.

Access Virtual, Public

Prototype
```
virtual Boolean CalcLocalFrameRect(
     Rect &outRect ) const;
```

Parameters The parameter for this method is:

`Rect&`	`outRect`	The Rect to calculate the pane's frame rectangle from.

Return Returns `true` if the frame is within QuickDraw space (16-bit), and returns `false` if the frame is outside QuickDraw space with `outRect` is unchanged.

CalcPortFrameRect()

Purpose Calculate the pane's frame rectangle in Port coordinates.

Returns true if the frame is within QuickDraw space (16-bit) Returns false if the frame is outside QuickDraw space and outRect is unchanged

Access Virtual, Public

Prototype
```
virtual Boolean CalcPortFrameRect(
     Rect &outRect ) const;
```

Parameters The parameter for this method is:

`Rect&`	`outRect`	The Rect to calculate the pane's frame rectangle from.

Return Returns `true` if the frame is within QuickDraw space (16-bit), and returns `false` if the frame is outside QuickDraw space with `outRect` is unchanged.

Click()

Purpose This is a wrapper method for handling a click inside a pane.

This method does some bookkeeping, executes Attachments, then calls ClickSelf() to actually respond to the click.

Access Virtual, Public

Prototype

```
virtual void Click(
     SMouseDownEvent &inMouseDown );
```

Parameters The parameter for this method is:

SMouseDownEvent&	inMouseDown	The mouse down event.

Return None

ClickSelf()

Purpose Respond to click inside this pane. You should override this to do something when the user clicks inside a pane.

Access Virtual, Public

Prototype

```
virtual void ClickSelf(
     const SMouseDownEvent& inMouseDown );
```

Parameters The parameter for this method is:

SMouseDownEvent&	inMouseDown	The mouse down event.

Return None

ClickTimesAreClose()

Purpose Indicate whether the time between the specified time and the time of the last mouse down are close enough to be considered a double-click.

Access Virtual, Public

Prototype

```
virtual Boolean ClickTimesAreClose(
     Uint32 inLastClickTime ) const;
```

Parameters The parameter for this method is:

Uint32	inLastClickTime	The mouse down event.

Return Returns `true` if a double-click, else `false`.

Contains()

Purpose Indicates whether a pane contains the specified point, which is in Port coordinates.

Note: This function just calls PointIsInFrame(), but is exists so that you can override it to change how the FindDeepSubPaneContaining() and FindShallowSubPaneContaining() search for Panes.

Access Virtual, Public

Prototype

```
virtual Boolean Contains(
     Int32 inHorizPort,
     Int32 inVertPort ) const;
```

Parameters The parameters for this method are:

`Int32`	`inHorizPort`	The horizontal coordinate.
`Int32`	`inVertPort`	The vertical coordinate.

Return Returns `true` if the pane contains the point, else returns `false`.

CountPanels()

Purpose Indicate the number of horizontal and vertical panels. A panel is a "frameful" of a pane.

Access Virtual, Public

Prototype

```
virtual void CountPanels(
    Uint32 &outHorizPanels,
    Uint32 &outVertPanels );
```

Parameters The parameters for this method are:

Uint32	outHorizPanels	The number of horizontal panels.
Uint32	outVertPanels	The number of vertical panels.

Return None

Deactivate()

Purpose Deactivate a pane.

Access Virtual, Public

Prototype

```
virtual void Deactivate();
```

Parameters None

Return None

DeactivateSelf()

Purpose This is called when a pane is being deactivated. You should override this for pane classes that change appearance when deactivated.

Access Virtual, Protected

Prototype
```
virtual void DeactivateSelf();
```

Parameters None

Return None

Disable()

Purpose Disable a pane.

Access Public

Prototype
```
virtual void Disable();
```

Parameters None

Return None

DisableSelf()

Purpose Pane is being disabled. You should override this for pane classes that change appearance when disabled. To avoid flicker, a pane can draw immediately from this method rather than refreshing and drawing at the next update event.

Access Virtual, Protected

Prototype
```
virtual void DisableSelf();
```

Parameters None

Return None

DontRefresh()

Purpose Validate the area occupied by a pane. This removes the pane area from the update region so that the pane won't be redrawn during the next update event.

Access Virtual, Public

Prototype

```
virtual void DontRefresh( Boolean inOKIfHidden );
```

Parameters The parameter for this method is:

`Boolean`	`inOKIfHidden`	Specifies whether to validate even if the pane is not visible. Pass true for this parameter to suppress the automatic refresh which occurs after hiding a pane.

Return None

Draw()

Purpose Try to draw the contents of a pane.

This is a wrapper function which calls DrawSelf() if it is proper for the pane to draw. This means that:

- pane is visible
- pane's frame is in QuickDraw space
- pane's frame intersects inSuperDrawRgnH
- pane can be focused

Access Virtual, Public

Prototype

```
virtual void Draw( RgnHandle inSuperDrawRgnH );
```

Parameters The parameter for this method is:

`RgnHandle`	`inSuperDrawRgnH`	Specifies, in Port coordinates, the portion of the pane's superview that needs to be drawn. Specify `nil` to bypass the intersection test.

Return None

DrawSelf()

Purpose This draws a box around the pane and a diagonal line from the top left to the bottom right corner using the default foreground and background colors for the pane.

On entry, the clipping region is the revealed area of the pane's superview. Therefore, it is possible for a pane to draw outside of its frame. You will not normally do this.

The port, coordinate system, and clipping region are set on entry. They must be the same upon exit. You are responsible for setting the Pen state and text characteristics to the proper values for your pane.

Access Virtual, Protected

Prototype
```
virtual void DrawSelf();
```

Parameters None

Return None

Enable()

Purpose Enable a pane.

Access Virtual, Public

Prototype
```
virtual void Enable();
```

Parameters None

Return None

EnableSelf()

Purpose Pane is being enabled. You should override this for pane classes that change appearance when enabled. To avoid flicker, a pane can draw immediately from this function rather than refreshing and drawing at the next update event.

Access Virtual, Protected

Prototype

```
virtual void EnableSelf();
```

Parameters None

Return None

EventMouseUp()

Purpose Respond to a mouse up event following a click (mouse down) inside a pane.

Access Virtual, Public

Prototype

```
virtual void EventMouseUp(
     const EventRecord &inMouseUp );
```

Parameters The parameter for this method is:

const EventRecord&	inMouseUp	The event info for the mouse up.

Return None

FindConstPaneByID()

Purpose Find the pane of a View which has the specified ID. This method searches all Panes contained within this View, not just direct subpanes. Returns `nil` if pane with the target ID is not found.

Use when you only want to inspect (and not change) the pane.

Access Virtual, Public

Prototype

```
virtual const LPane* FindConstPaneByID(
     PaneIDT inPaneID ) const;
```

Parameters The parameter for this method is:

`PaneIDT`	`inPaneID`	The resource ID for the pane.

Return A pointer to the pane.

FindDeepSubPaneContaining()

Purpose Return the most deeply nested subpane which contains the specified point, which is in Port coordinates.

Access Virtual, Public

Prototype

```
virtual LPane* FindDeepSubPaneContaining(
     Int32 inHorizPort,
     Int32 inVertPort ) const;
```

Parameters The parameters for this method are:

Int32	inHorizPort	The horizontal coordinate.
Int32	inVertPort	The vertical coordinate.

Return Return `nil` if no subpane contains the point, else returns a pointer to the pane.

FindPaneByID()

Purpose Find the pane of a View which has the specified ID. This method searches all Panes contained within this View, not just direct subpanes.

Access Virtual, Public

Prototype

```
virtual LPane* FindPaneByID( PaneIDT inPaneID );
```

Parameters The parameter for this method is:

`PaneIDT`	`inPaneID`	The resource ID for the pane.

Return Returns `nil` if pane with the target ID is not found.

FindShallowSubPaneContaining()

Purpose Return the immediate subpane which contains the specified point, which is in Port coordinates.

Same as FindSubPaneHitBy() except that it tests using Contains() rather than IsHitBy(). `Contains()` does not check if the pane is enabled.

Access Virtual, Public

Prototype

```
virtual LPane* FindShallowSubPaneContaining(
     Int32 inHorizPort,
     Int32 inVertPort ) const;
```

Parameters The parameters for this method are:

Int32	inHorizPort	The horizontal coordinate.
Int32	inVertPort	The vertical coordinate.

Return Return `nil` if no subpane contains the point, else returns a pointer to the pane.

FindSubPaneHitBy()

Purpose Find the subpane of this pane that is hit by the specified point.

Access Virtual, Public

Prototype

```
virtual LPane* FindSubPaneHitBy(
     Int32 inHorizPort,
     Int32 inVertPort ) const;
```

Parameters The parameters for this method are:

`Int32`	`inHorizPort`	The horizontal coordinate.
`Int32`	`inVertPort`	The vertical coordinate.

Return Return nil if no subpane is hit, else return a pointer to the pane.

Remarks `inHorizPort` and `inVertPort` are in Port coordinates.

FinishCreate()

Purpose Wrapper function for FinishCreateSelf(). You will rarely want to override this function.

Subpanes are told to `FinishCreateSelf` *before* their superview. Therefore, a superview is assured that all its subpanes have finished creating when its `FinishCreateSelf` function gets called.

Access Virtual, Public

Prototype

```
virtual void FinishCreate();
```

Parameters None

Return None

FinishCreateSelf()

Purpose Override this method to perform finishing touches that depend on the entire pane hierarchy being constructed.

For example, if a View wants to store a pointer to a subpane, it should override this function to call FindPaneByID() for that pane. This saves the overhead of repeatedly calling `FindPaneByID()` when the View wants to access that subpane. You can't do this from a constructor because subpanes are created after their superview.

Access Virtual, Protected

Prototype

```
virtual void FinishCreateSelf();
```

Parameters None

Return None

FocusDraw()

Purpose Prepare for drawing in the pane. Throws an exception if the pane has no superview. A pane does not have its own coordinate system and clipping region.

This method relies on its superview to set the focus. Even if FocusDraw() returns `false`, the pane's GrafPort and clipping region of its superview will be set (unless pane has no superview)

Access Virtual, Public

Prototype

```
virtual Boolean FocusDraw( LPane* inSubPane );
```

Parameters The parameter for this method is:

`LPane*`	`inSubPane`	The pointer to the Subpane.

Return Returns `true` if superview is focused.

FocusExposed()

Purpose Prepare for drawing in the pane.

A pane is exposed when all the following conditions are true:

- pane's "visible" flag is on
- pane has a superview
- superview is exposed
- pane's frame intersects exposed area of its superview

Since a pane does not have its own coordinate system, focusing a pane sets up the port, coordinate system, and clipping region of its superview.

Access Virtual, Public

Prototype

```
virtual Boolean FocusExposed(
     Boolean inAlwaysFocus );
```

Parameters The parameter for this method is:

`Boolean`	`inAlwaysFocus`	Specifies whether to focus the pane even if it is not exposed. pane is not focused if it is not exposed *and* `inAlwaysFocus` is false.

Return Returns `true` if a View is exposed, meaning that it is visible and that it is revealed through the Frames of all its SuperViews.

GetActiveState()

Purpose Indicate the state of the pane, stored in the mActive data member.

Access Public

Prototype

```
ETriState GetActiveState() const;
```

Parameters None

Return The state of the pane.

GetClickCount()

Purpose Indicate the value of the sClickCount data member.

Access Inline, Static, Public

Prototype
```
static Int16 GetClickCount();
```

Parameters None

Return The value of sClickCount.

GetDefaultView()

Purpose Returns the value of sDefaultView.

Access Inline, Static, Public

Prototype
```
static LView* GetDefaultView();
```

Parameters None

Return A pointer to the LView.

GetDescriptor()

Purpose Returns the descriptor string for a pane.

Access Virtual, Public

Prototype
```
virtual StringPtr GetDescriptor(
     Str255 outDescriptor ) const;
```

Parameters The parameter for this method is:

Str255	outDescriptor	The descriptor string.

Return A pointer to the descriptor string.

GetEnabledState()

Purpose Returns the value of the mEnabled data member.

Access Public

Prototype
```
ETriState GetEnabledState() const;
```

Parameters None

Return Returns the value of mEnabled.

GetFrameBinding()

Purpose Retrieves the value of mFrameBinding.

Access Public

Prototype
```
void GetFrameBinding(
     SBooleanRect &outFrameBinding ) const;
```

Parameters The parameter for this method is:

`SBooleanRect&`	`outFrameBinding`	The frame Rect.

Return None

GetFrameLocation()

Purpose Get the location of a pane's frame, in 32-bit Port coordinates.

Access Public

Prototype
```
void GetFrameLocation(
     SPoint32 &outLocation ) const;
```

Parameters The parameter for this method is:

`SPoint32&`	`outLocation`	The location of the frame.

Return None

GetFrameSize()

Purpose Get the width and height, in pixels, of the frame of a pane

Access Public

Prototype
```
void GetFrameSize(
     SDimension16 &outSize ) const;
```

Parameters The parameter for this method is:

`SDimension16&`	`outSize`	The size of the frame.

Return None

GetLastPaneClicked()

Purpose Returns the value of the sLastPaneClicked data member.

Access Static, Public

Prototype
```
static LPane* GetLastPaneClicked();
```

Parameters None

Return A pointer to the last pane clicked.

GetLastPaneMoused()

Purpose Returns the value of the sLastPaneMoused data member.

Access Static, Public

Prototype `static LPane* GetLastPaneMoused();`

Parameters None

Return A pointer to the last pane moused.

GetLocalUpdateRgn()

Purpose Return the region being updated. The region will be empty if called outside of a Draw() operation

Access Virtual, Public

Prototype `virtual RgnHandle GetLocalUpdateRgn();`

Parameters None

Return A handle the the region.

Remarks This region is non-empty only within the scope of a `Draw()` operation. Call this function from a DrawSelf() function if you want to know the actual area that requires redrawing. Most of the time, you will just draw everything and let the clipping region mask out unnecessary drawing. However, if your drawing is complex, your program might run faster if you only draw the portion which is inside the update region.

GetMacPort()

Purpose Return the Mac OS GrafPort.

Access Virtual, Public

Prototype `virtual GrafPtr GetMacPort() const;`

Parameters None

Return A pointer to the Mac OS GrafPort.

GetPaneID()

Purpose Returns the value of the mPaneID data member.

Access Public

Prototype
```
PaneIDT GetPaneID() const;
```

Parameters None

Return The value of mPaneID.

GetSuperView()

Purpose Returns the value of mSuperView.

Access Public

Prototype
```
LView* GetSuperView() const;
```

Parameters None

Return A pointer to the LView held in mSuperView.

GetUserCon()

Purpose Return the value of mUserCon.

Access Virtual, Public

Prototype
```
virtual Int32 GetUserCon() const;
```

Parameters None

Return Return the value of mUserCon.

GetValue()

Purpose Returns the value for the pane. You should override this if you want this method to return something other than 0.

Access Virtual, Public

Prototype
```
virtual Int32 GetValue() const;
```

Parameters None

Return The pane value.

GetVisibleState()

Purpose Return the value of the mVisible data member.

Access Public

Prototype
```
ETriState GetVisibleState() const;
```

Parameters None

Return The value of mVisible.

GlobalToPortPoint()

Purpose Convert a global point to a Port point.

Access Virtual, Public

Prototype
```
virtual void GlobalToPortPoint(
     Point &ioPoint ) const;
```

Parameters The parameter for this method is:

`Point&`	`ioPoint`	The Point.

Return None

Hide()

Purpose Make a pane invisible.

Access Virtual, Public

Prototype
```
virtual void Hide();
```

Parameters None

Return None

HideSelf()

Purpose Pane is being made invisible. You should override this method for pane classes that need to be informed when they are made invisible

Access Virtual, Protected

Prototype
```
virtual void HideSelf();
```

Parameters None

Return None

InitPane()

Purpose Initializer for the pane.

Access Protected

Prototype
```
void InitPane( const SPaneInfo &inPaneInfo );
```

Parameters The parameter for this method is:

`const SPaneInfo&`	`inPaneInfo`	The Superpane info.

Return None

InvalPortRect()

Purpose Invalidate the Port rectangle.

All rectangles and regions must be in Port coordinates.

Access Virtual, Public

Prototype

```
virtual void InvalPortRect( const Rect *inRect );
```

Parameters The parameter for this method is:

const Rect *	inRect	The Rect.

Return None

Remarks You should use these routines rather than the Toolbox traps `InvalRect`, `InvalRgn`, `ValidRect`, and `ValidRgn`. Those traps require that the current GrafPort be a Window. However, a pane could be in another kind of GrafPort, such as a Printer Port or GWorld, where calling one of those traps would cause a crash (when the Toolbox tries to access a nonexistant update region).

InvalPortRgn()

Purpose Invalidate the Port region.

Access Virtual, Public

Prototype

```
virtual void InvalPortRgn(
     RgnHandle inRgnH );
```

Parameters The parameter for this method is:

RgnHandle	inRgnH	The region handle.

Return None

IsActive()

Purpose Indicates whether a pane is active. A pane is "active" if its active flag is on and it is visible.

Access Public

Prototype
```
Boolean IsActive() const;
```

Parameters None

Return Returns `true` if the pane is active, else `false`.

IsAreaInQDSpace()

Purpose Indicates whether an area is inQuickDraw space.

Access Static, Public

Prototype
```
static Boolean IsAreaInQDSpace(
     Int32 inLeft,
     Int32 inTop,
     Int16 inWidth,
     Int16 inHeight );
```

Parameters The parameters for this method are:

`Int32`	inLeft	The left coordinate.
`Int32`	inTop	The top coordinate.
Int16	inWidth	The width.
Int16	inHeight	The height.

Return Returns `true` if the area is in the space, else `false`.

IsEnabled()

Purpose Return whether a pane is enabled. A pane is "enabled" if its enabled flag is on and it is visible.

Access Public

Prototype
```
Boolean IsEnabled() const;
```

Parameters None

Return Returns `true` if the pane is enabled, else returns `false`.

IsHitBy()

Purpose Indicates whether a pane is hit by the specified point, which is in Port coordinates. A pane is hit if the point is inside its frame and the pane is enabled.

Access Virtual, Public

Prototype
```
virtual Boolean IsHitBy(
     Int32 inHorizPort,
     Int32 inVertPort );
```

Parameters The parameters for this method are:

`Int32`	`inHorizPort`	The horizontal coordinate.
`Int32`	`inVertPort`	The vertical coordinate.

Return Returns `true` if the pane is hit by the specified point, else returns `false`.

IsVisible()

Purpose Indicates whether a pane is visible. A pane is "visible" if its visible flag is on.

Access Public

Prototype

```
Boolean IsVisible() const;
```

Parameters None

Return Returns `true` if the pane is visible.

LocalToPortPoint()

Purpose Convert point from Local to Port coordinates.

Access Virtual, Public

Prototype

```
virtual void LocalToPortPoint(
     Point &ioPoint ) const;
```

Parameters The parameter for this method is:

Point &	ioPoint	The point to convert (conversion is in place).

Return None

MouseEnter()

Purpose This method could be overrided to provide useful functionality when the mouse pointer enters the pane.

Access Virtual, Public

Prototype

```
virtual void MouseEnter(
     Point inPortPt,
     const EventRecord& inMacEvent );
```

Parameters The parameter for this method is:

const EventRecord&	inMouseUp	The event info for the mouse up.

Return None

MouseLeave()

Purpose This method could be overrided to provide useful functionality when the mouse pointer leaves the pane.

Access Virtual, Public

Prototype

```
virtual void MouseLeave();
```

Parameters None

Return None

MouseWithin()

Purpose Override this method to provide useful functionality when the mouse pointer is within the pane.

Access Virtual, Public

Prototype

```
virtual void MouseWithin(
     Point inPortPt,
     const EventRecord& inMacEvent );
```

Parameters The parameters for this method are:

`Point`	`inPortPt`	The port point.
`const EventRecord&`	`inMacEvent`	The event info.

Return None

MoveBy()

Purpose Move the location of the frame by the specified amounts. This specifies, in pixels, how far to move the frame (within its surrounding Image).

`inHorizDelta` and `inVertDelta` specify, in pixels, how far to move the frame (within its surrounding Image).

Access Virtual, Public

Prototype

```
virtual void MoveBy(
     Int32 inHorizDelta,
     Int32 inVertDelta,
     Boolean inRefresh );
```

Parameters The parameters for this method are:

Int32	inHorizDelta	Positive horizontal deltas move to the right, negative to the left.
Int32	inVertDelta	Positive vertical deltas move down, negative up.
Boolean	inRefresh	Indicates whether to refresh or not.

Return None

PlaceInSuperFrameAt()

Purpose Place the pane at a location relative to the frame of its superview.

Access Public

Prototype

```
void PlaceInSuperFrameAt(
     Int32 inHorizOffset,
     Int32 inVertOffset,
     Boolean inRefresh);
```

Parameters The parameters for this method are:

`Int32`	`inHorizOffset`	Specifies, in pixels, how far the left edge of the frame is from the left edge of its parent frame. Positive offsets are to the left, negative to the right.
`Int32`	`inVertOffset`	Specifies, in pixels, how far the top edge of the frame is from the top edge of its parent frame. Positive offsets are below, negative above.
`Boolean`	`inRefresh`	This value is passed to MoveBy().

Return None

PlaceInSuperImageAt()

Purpose Place the pane within the Image of its superview.

Access Virtual, Public

Prototype

```
virtual void PlaceInSuperImageAt(
     Int32 inHorizOffset,
     Int32 inVertOffset,
     Boolean inRefresh );
```

Parameters The parameters for this method are:

`Int32`	`inHorizOffset`	Specify the distance of the top left of the frame from the top left of the Image of its superview.
`Int32`	`inVertOffset`	Specify the distance of the top left of the frame from the top left of the Image of its superview.
`Boolean`	`inRefresh`	This value is passed to MoveBy().

Return None

PointIsInFrame()

Purpose This method determines whether a given coordinate is within the perimeter of the button.

Access Virtual, Public

Prototype

```
virtual Boolean PointIsInFrame( Int32 inHoriz,
     Int32 inVert ) const;
```

Parameters This method has the following parameters:

Int32	inHoriz	The horizontal coordinate
Int32	inVert	The vertical coordinate

Return Boolean of `true` if the point is within the perimeter, `false` otherwise.

PointsAreClose()

Purpose Return whether the two points are close enough to be part of a multi-click. Points are in Local coordinates.

Access Virtual, Public

Prototype

```
virtual Boolean PointsAreClose(
     Point inFirstPt,
     Point inSecondPt ) const;
```

Parameters This method has the following parameters:

`Point`	`inFirstPt`	The first point.
`Point`	`inSecondPt`	The second point.

Return Returns `true` if the points are close enough to be a multi-click, else `false`.

PortToGlobalPoint()

Purpose Convert a point to global coordinates.

Access Virtual, Public

Prototype

```
virtual void PortToGlobalPoint(
     Point &ioPoint ) const;
```

Parameters The parameter for this method is:

Point&	ioPoint	The point to convert.

Return None

PortToLocalPoint()

Purpose Convert point from Port to Local coordinates.

Access Virtual, Public

Prototype

```
virtual void PortToLocalPoint(
     Point &ioPoint ) const;
```

Parameters The parameter for this method is:

`Point&`	`ioPoint`	The point to convert.

Return None

PrintPanel()

Purpose Try to print a panel of a pane. Since a pane does not scroll, it just prints itself for every panel.

This is a wrapper function which calls PrintPanelSelf() if it is proper for the pane to print. This means that:

- pane is visible
- pane's frame is in QuickDraw space
- pane's frame intersects inSuperDrawRgnH
- pane can be focused

Access Virtual, Public

Prototype

```
virtual void PrintPanel(
     const PanelSpec &inPanel,
     RgnHandle inSuperPrintRgnH );
```

Parameters The parameters for this method are:

`const PanelSpec&`	`inPanel`	The panel info.
`RgnHandle`	`inSuperPrintRgnH`	Specifies, in Port coordinates, the portion of the pane's superview that needs to be printed. Specify `nil` to bypass the intersection test.

Return None

PrintPanelSelf()

Purpose Print a panel of a pane.

Access VIrtual, Public

Prototype

```
virtual void PrintPanelSelf(
     const PanelSpec& inPanel);
```

Parameters The parameter for this method is:

`const PanelSpec&`	`inPanel`	The panel info.

Return None

PutInside()

Purpose Put pane inside the specified View. Location is unspecified.

InitPane() calls PutInside() with `inOrient` set to `false` since the pane is not fully constructed at that point, meaning that any functions called by OrientSubPane() will be the ones for LPane, and not any overrides. This may result in the pane not being properly oriented. After creating a pane, you should call FinishCreate(), which will call OrientSubPane() for the fully constructed pane object.

Access Public

Prototype

```
void PutInside( LView *inView, Boolean inOrient );
```

Parameters The parameters for this method are:

`Boolean`	`inOrient`	Specifies whether or not to orient the pane within its new superview. The default value is `true`, which is what you'll normally want.
`LView*`	`inView`	The pointer to the View.

Return None

Refresh()

Purpose Invalidate the area occupied by a pane. This forces an update event that, when processed, will redraw the pane. This method does nothing if the pane is not exposed.

Access Public

Prototype

```
virtual void Refresh();
```

Parameters None

Return None

ResizeFrameBy()

Purpose Change the frame size by the specified amounts.

`inWidthDelta` and `inHeightDelta` specify, in pixels, how much larger to make the frame. Positive deltas increase the size, negative deltas reduce the size.

Access Virtual, Public

Prototype

```
virtual void ResizeFrameBy(
     Int16 inWidthDelta,
     Int16 inHeightDelta,
     Boolean inRefresh );
```

Parameters The parameters for this method are:

`Int32`	`inWidthDelta`	The width change.
`Int32`	`inHeightDelta`	The height change.
`Boolean`	`inRefresh`	Indicates whether to refresh or not.

Return None

Remarks Note that this function only changes the frame size, not the bounds of any enclosing view. To change the actual size (on screen) of an LWindow object, call ResizeWindowBy() or ResizeWindowTo().

ResizeFrameTo()

Purpose Set the dimensions of the frame.

`inWidthDelta` and `inHeightDelta` specify, in pixels, how much larger to make the frame. Positive deltas increase the size, negative deltas reduce the size.

Access Public

Prototype

```
void ResizeFrameTo(
     Int16 inWidth,
     Int16 inHeight,
     Boolean inRefresh );
```

Parameters The parameters for this method are:

`Int16`	`inWidth`	The horizontal width change to resize the window by.
`Int16`	`inHeight`	The vertical height change to resize the window by.
`Boolean`	`inRefresh`	Indicates whether to refresh or not.

Return None

RestorePlace()

Purpose Read size and location information stored in a Stream by the SavePlace() function

Access Virtual, Public

Prototype `virtual void RestorePlace( LStream *inPlace );`

Parameters The parameter for this method is:

`LStream*`	`inPlace`	The stream to read from.

Return None

SavePlace()

Purpose Write size and location information to a Stream for later retrieval by the RestorePlace() function.

Access Virtual, Public

Prototype
```
virtual void SavePlace( LStream *outPlace );
```

Parameters The parameter for this method is:

`LStream*`	`outPlace`	The stream to write to.

Return None

ScrollToPanel()

Purpose Scroll pane to the specified panel. Panes do not scroll, so just return `true` indicating that the panel is always valid

Access Virtual, Public

Prototype
```
virtual Boolean ScrollToPanel(
     const PanelSpec &inPanel );
```

Parameters The parameter for this method is:

`const PanelSpec&`	`inPanel`	The panel info.

Return Return whether the specified panel exists. If it doesn't, View is not scrolled.

SetDefaultView()

Purpose Return the value of the sDefaultView data member.

Access Static, Public

Prototype
```
static void SetDefaultView( LView *inView);
```

Parameters None

Return Return the value of the sDefaultView.

SetDescriptor()

Purpose Set the descriptor string.

Access Virtual, Public

Prototype
```
virtual void SetDescriptor(
     ConstStringPtr inDescriptor );
```

Parameters The parameter for this method is:

`ConstStringPtr`	`inDescriptor`	The string pointer to set for the descriptor.

Return None

SetForeAndBackColors()

Purpose Specify the foreground and/or background colors of a Window. Specify `nil` for `inForeColor` and/or `inBackColor` to leave that color trait unchanged

Access Virtual, Public

Prototype

```
virtual void SetForeAndBackColors(
     const RGBColor *inForeColor,
     const RGBColor *inBackColor );
```

Parameters The parameters for this method are:

`const RGBColor*`	`inForeColor`	The foreground color.
`const RGBColor*`	`inBackColor`	The background color.

Return None

SetFrameBinding()

Purpose This method sets the value of the mFrameBinding data member.

Access Public

Prototype

```
void SetFrameBinding(
     const SBooleanRect &inFrameBinding );
```

Parameters The parameter for this method is:

`const SBooleanRect &`	`inFrameBinding`	The rect for the frame.

Return None

SetLastPaneMoused()

Purpose This data member sets the value of the sLastPaneMoused data member.

Access Static, Public

Prototype `static void SetLastPaneMoused( LPane *inPane );`

Parameters The parameter for this method is:

`LPane*`	`inPane`	The rect for the frame.

Return None

SetPaneID()

Purpose Set the value of the mPaneID data member.

Access Public

Prototype `void SetPaneID( PaneIDT inPaneID );`

Parameters The parameter for this method is:

`PaneIDT`	`inPaneID`	The resource ID for the pane.

Return None

SetRefreshAllWhenResized()

Purpose This method sets the value of the mRefreshAllWhenResized data member.

Access Public

Prototype
```
void SetRefreshAllWhenResized(
     Boolean inRefreshAll );
```

Parameters The parameter for this method is:

`Boolean`	`inRefreshAll`	Whether to refresh all panes when resized.

Return None

SetUserCon()

Purpose Set the mUserCon data member.

Access Virtual, Public

Prototype
```
virtual void SetUserCon( Int32 inUserCon );
```

Parameters The parameter for this method is:

`Int32`	`inUserCon`	The value to set for the data member.

Return None

SetValue()

Purpose Override this method to provide useful functionality for setting the value of a pane.

Access Virtual, Public

Prototype
```
virtual void SetValue( Int32 inValue );
```

Parameters The parameter for this method is:

`Int32`	`inValue`	The value to set.

Return None

Show()

Purpose Make a pane visible.

Access Virtual, Public

Prototype `virtual void Show();`

Parameters None

Return None

ShowSelf()

Purpose Pane is being made visible. You should override this method for pane classes that need to be informed when they are made visible.

Access Virtual, Protected

Prototype `virtual void ShowSelf();`

Parameters None

Return None

SuperActivate()

Purpose The superview of a pane has been activated.

Access Virtual, Protected

Prototype `virtual void SuperActivate();`

Parameters None

Return None

SuperDeactivate()

Purpose The superview of a pane has been deactivated.

Access Virtual, Protected

Prototype `virtual void SuperDeactivate();`

Parameters None

Return None

SuperDisable()

Purpose The superview of a pane has been disabled.

Access Virtual, Protected

Prototype
```
virtual void SuperDisable();
```

Parameters None

Return None

SuperEnable()

Purpose The superview of a pane has been enabled.

Access Virtual, Protected

Prototype
```
virtual void SuperEnable();
```

Parameters None

Return None

SuperHide()

Purpose The superview of a pane has been hidden.

Access Virtual, Protected

Prototype
```
virtual void SuperHide();
```

Parameters None

Return None

SuperPrintPanel()

Purpose superview is printing a panel. The View is not in control of pagination. In general, it is not clear how to print nested scrolling views. This function just prints the View at its current location, without scrolling to a particular panel.

Access Virtual, Public

Prototype

```
virtual void SuperPrintPanel(
     const PanelSpec &inSuperPanel,
     RgnHandle inSuperPrintRgnH );
```

Parameters The parameters for this method are:

`const PanelSpec&`	inSuperPanel	The panel info.
`RgnHandle`	`inSuperPrintRgnH`	Specifies, in Port coordinates, the portion of the pane's superview that needs to be printed. Specify `nil` to bypass the intersection test.

Return None

SuperShow()

Purpose The superview of a pane has become visible.

Access Virtual, Protected

Prototype

```
virtual void SuperShow();
```

Parameters None

Return None

UpdateClickCount()

Purpose Determine if the mouse down is part of a multi-click and set internal counters.

Access Virtual, Public

Prototype

```
virtual void UpdateClickCount(
     const SMouseDownEvent &inMouseDown );
```

Parameters The parameter for this method is:

`const EventRecord&`	`inMouseDown`	The event info.

Return None

UpdatePort()

Purpose Redraw invalidated area of the Port containing the pane.

For Panes that are in a Window Port (i.e., the ultimate super view is a Window), this forces an immediate redraw of the update region of the Window. Since this message is really directed at the Port containing the pane (rather than the pane itself), the update occurs even if the pane is not visible.

Panes that maintain a Mac GrafPort must override this function if they support updating.

Access Virtual, Public

Prototype

```
virtual void UpdatePort();
```

Parameters None

Return None

ValidPortRect()

Purpose Sets the port rect.

Access Virtual, Public

Prototype
```
virtual void ValidPortRect( const Rect *inRect );
```

Parameters The parameter for this method is:

`const Rect*`	`inRect`	The Rect.

Return None

ValidPortRgn()

Purpose Sets the port region.

Access Virtual, Public

Prototype
```
virtual void ValidPortRgn( RgnHandle inRgnH );
```

Parameters The parameter for this method is:

`RgnHandle`	`inRgnH`	The region handle.

Return None

sDefaultView

Purpose The default view pointer storage.

Access Static, Protected

Prototype
```
static LView *sDefaultView;
```

sLastPaneClicked

Purpose The storage for the last pane clicked.

Access Static, Protected

Prototype
```
static LPane *sLastPaneClicked;
```

sLastPaneMoused

Purpose The storage for the last pane moused.

Access Static, Protected

Prototype
```
static LPane *sLastPaneMoused;
```

sWhenLastMouseUp

Purpose Storage for when the last mouse up occurred.

Access Static, Protected

Prototype
```
static Uint32 sWhenLastMouseUp
```

sWhenLastMouseDown

Purpose Storage for when the last mouse down occurred.

Access Static, Protected

Prototype
```
static Uint32 sWhenLastMouseDown;
```

sWhereLastMouseDown

Purpose Storage for where the last mouse down occurred.

Access Static, Protected

Prototype `static Point sWhereLastMouseDown;`

sClickCount

Purpose Storage for the click count.

Access Static, Protected

Prototype `static Int16 sClickCount;`

mPaneID

Purpose The storage for the pane ID.

Access Protected

Prototype `PaneIDT mPaneID;`

mFrameSize

Purpose The storage for the frame size.

Access Protected

Prototype `SDimension16 mFrameSize;`

mFrameLocation

Purpose Storage for the frame location.

Access Protected

Prototype `SPoint32 mFrameLocation;`

mFrameBinding

Purpose Storage for the frame binding.

Access Protected

Prototype
```
SBooleanRect mFrameBinding;
```

mUserCon

Purpose Storage for the UserCon storage area.

Access Protected

Prototype
```
Int32 mUserCon;
```

mSuperView

Purpose Storage for the superview.

Access Protected

Prototype
```
LView *mSuperView;
```

mVisible

Purpose Storage for whether the pane is visible or not.

Access Protected

Prototype
```
ETriState mVisible;
```

mActive

Purpose This is storage for whether the pane is active or not.

Access Protected

Prototype
```
ETriState mActive;
```

mEnabled

Purpose This is storage for whether the pane is enabled or not.

Access Protected

Prototype

```
ETriState mEnabled;
```

mRefreshAllWhenResized

Purpose Storage for whether to resize all when the pane is resized.

Access Protected

Prototype

```
Boolean mRefreshAllWhenResized;
```

LPeriodical

Overview

LPeriodical is an abstract PowerPlant base class that is multiply inherited by other framework objects. It is used for giving objects the ability to do work at periodic intervals. In PowerPlant, an object that receives processor time on a periodic basis is said to be a Periodical.

Since LPeriodical is an abstract base class, you can't instantiate an LPeriodical object. Instead, you inherit from LPeriodical in your object's class definition, and define any overrides of the base class methods needed by your object.

This class maintains two static queues, an Idler queue and a Repeater queue. It is up to the caller to determine the meaning of each queue and when to devote time to the Periodicals in them. The PowerPlant LApplication class devotes time to Idlers at Null Event time, and devotes time to Repeaters after every event.

Methods

The methods in this class are:

LPeriodical()	~LPeriodical()
DeleteIdlerAndRepeaterQueues()	DevoteTimeToIdlers()
DevoteTimeToRepeaters()	SpendTime()
StartIdling()	StartRepeating()
StopIdling()	StopRepeating()

Data Members

The data members in this class are:

sIdlerQ	sRepeaterQ

Operation

Many different objects in your program may require processor time on a periodic basis. For example, changing the text edit insertion caret in LEditField requires that some periodic operations are performed. Another example is LGrowZone(), which requires periodic polling of memory conditions in order to be effective at handling low-memory situations.

There are two different kinds of periodical objects that you create:

- Repeaters
- Idlers

An Idler object is an object that gets processing time when the application gets a null event. In other words, when the application thread has nothing else to do, it will process the Idler queue.

A Repeater object is an object that gets processing time after every event.

Source files (Feature Classes)

`LPeriodical.h`

`LPeriodical.cp`

See also LArray

TArray

LPeriodical()

Purpose The constructor for LPeriodical doesn't do anything.

Access Public

Prototype

```
LPeriodical();
```

Parameters None

Return None

~LPeriodical()

Purpose The destructor for LPeriodical stops the Idler and Repeater queue processing.

Access Public

Prototype `virtual ~LPeriodical();`

Parameters None

Return None

DeleteIdlerAndRepeaterQueues()

Purpose This method deletes the internal queues used to store pointers to Idler and Repeater objects. This method does not delete the Idler and Repeater objects themselves.

You don't need to call this when quitting an application, since the queues will disappear when the System deallocates the application heap. However, you may want to call this if you like deleting every object that is created via the `new` operator.

Access Public, Static

Prototype
```
static void DeleteIdlerAndRepeaterQueues();
```

Parameters None

Return None

Remarks Normally, you will only use this routine to clean up memory when terminating a code resource or fragment. You do not normally need to call this method in from application code.

DevoteTimeToIdlers()

Purpose This method calls SpendTime() for each Periodical object in the Idler queue. This gives each object some procesing time.

Access Public, Static

Prototype
```
static void DevoteTimeToIdlers(
     const EventRecord &inMacEvent);
```

Parameters This method has the following parameter:

`const EventRecord&`	`inMacEvent`	A struct containing the parameters from the last event that occurred.

Return None

DevoteTimeToRepeaters()

Purpose This method calls SpendTime() for each Periodical object in the Repeater queue. This gives each object some procesing time.

Access Public, Static

Prototype

```
static void DevoteTimeToRepeaters(
     const EventRecord &inMacEvent);
```

Parameters This method has the following parameter:

`const EventRecord&`	`inMacEvent`	A struct containing the parameters from the last event that occurred.

Return None

SpendTime()

Purpose This method is a pure virtual method. You must override it in your concrete classes. You provide code that implements the operations that are required for your object to perform when it is given processing time.

Access Virtual, Public

Prototype

```
virtual void SpendTime(
      const EventRecord &inMacEvent) = 0;
```

Parameters This method has the following parameter:

`const EventRecord&`	`inMacEvent`	A struct containing the parameters from the last event that occurred.

Return None

StartIdling()

Purpose This method creates an Idler queue if one doesn't already exist. Then, it adds this Periodical object to the end of the queue if it isn't already in there.

Access Public, Virtual

Prototype
```
virtual void StartIdling();
```

Parameters None

Return None

StartRepeating()

Purpose This method creates a Repeater queue if one doesn't already exist. Then, it adds this Periodical object to the end of the queue if it isn't already in there.

Access Public, Virtual

Prototype
```
virtual void StartRepeating();
```

Parameters None

Return None

StopIdling()

Purpose Remove this Periodical object from the Idler queue.

Access Public, Virtual

Prototype
```
virtual void StopIdling();
```

Parameters None

Return None

StopRepeating()

Purpose Remove this Periodical object from the Repeater queue.

Access Public, Virtual

Prototype
```
virtual void StopRepeating();
```

Parameters None

Return None

sIdlerQ

Purpose This data member is a pointer to a TArray of pointers to LPeriodical objects. In short, this data member is a pointer to a list of objects in the Idler queue.

Access Static, Protected

Prototype
```
static TArray<LPeriodical*> *sIdlerQ;
```

Remarks In general, you will want to use the methods in the LPeriodical class to manipulate the sIdlerQ list, rather than accessing it directly.

sRepeaterQ

Purpose This data member is a pointer to a TArray of pointers to LPeriodical objects. In short, this data member is a pointer to a list of objects in the Repeater queue.

Access Static, Protected

Prototype

```
static TArray<LPeriodical*> *sRepeaterQ;
```

Remarks In general, you will want to use the methods in the LPeriodical class to manipulate the sRepeaterQ list, rather than accessing it directly.

LPicture

Overview LPicture is a PowerPlant class that is used for displaying `PICT` resources.

Methods The methods in this class are:

LPicture() DrawSelf()

GetPictureID() InitPicture()

SetPictureID()

Data Members The data members in this class are:

mPICTid

Operation This is a simple extension of the LView class.

Source files Pane Classes)

`LPicture.h`

`LPicture.cp`

Ancestors LAttachable

LPane

LView

LPicture()

Purpose The constructors create objects from the passed-in parameters.

Access Public

Prototype

```
LPicture();
LPicture( const LPicture &inOriginal );
```

```
LPicture( const SPaneInfo &inPaneInfo,
     const SViewInfo &inViewInfo,
     ResIDTinPICTid );
LPicture( LStream *inStream );
LPicture( ResIDT inPictureID );
```

Parameters The parameters for these constructors are:

`const LPicture&`	`inOriginal`	A reference to the picture to copy.
`const SPaneInfo&`	`inPaneInfo`	The Superpane info.
`const SViewInfo&`	`inViewInfo`	The Superview info.
`ResIDT`	`inPICTid`	The resource ID for the `PICT` resource.
`LStream*`	`inStream`	The stream to read from.
`ResIDT`	`inPictureID`	The resource ID for the `PICT` resource.

DrawSelf()

Purpose Draw the picture. This is an override of DrawSelf() in LPane.

GetPictureID()

Purpose Retrieve the picture ID.

Access Public

Prototype

```
ResIDT GetPictureID() const;
```

Parameters None

Return A resource ID for the picture.

InitPicture()

Purpose Private Initializer. Assumes mPICTid is set.

Access Private

Prototype
```
void InitPicture();
```

Parameters None

Return None

SetPictureID()

Purpose Set the PICT Resource ID associated with a picture. This method changes the size of the image to match the bounds of the `PICT`.

Access Public

Prototype
```
void SetPictureID( ResIDT inPictureID );
```

Parameters This method has the following parameter:

`ResIDT`	`inPictureID`	The resource ID for the `PICT` resource.

Return None

mPICTid

Purpose The picture resource ID.

Access Protected

Prototype
```
ResIDT mPICTid;
```

LPlaceHolder

Overview

LPlaceHolder is a PowerPlant class that is designed to assist the printing process. The purpose of a placeholder is to allow you to print a pane at a size and/or location that is different from the pane's characteristics when displayed in a window.

Methods

The methods in this class are:

LPlaceHolder()	~LPlaceHolder()
CountPanels()	InstallOccupant()
RemoveOccupant()	ScrollToPanel()

Data Members

The data members in this class are:

mOccupant	mOccupantSuperView
mOccupantPlaceH	mOccupantAlignment

Operation

A placeholder has two characteristics that differentiate it from other views. It has an occupant, and the occupant has an alignment. For more details on this class, refer to *The PowerPlant Book*.

Source files

(Pane Classes)

`LPlaceHolder.h`

`LPlaceHolder.cp`

Ancestors

LAttachable

LPane

LView

LPlaceHolder()

Purpose

The constructor creates objects from the passed-in parameters.

Access Public

Prototype

```
LPlaceHolder();
LPlaceHolder( const LPlaceHolder &inOriginal );
LPlaceHolder( const SPaneInfo &inPaneInfo,
     const SViewInfo &inViewInfo,
     Int16 inOccupantAlignment );
LPlaceHolder( LStream *inStream );
```

Parameters The constructors take the following parameters:

`const LPlaceHolder&`	`inOriginal`	The object to copy.
`const SPaneInfo&`	`inPaneInfo`	The Superpane.
`const SViewInfo&`	`inViewInfo`	The Superview.
`Int16`	`inOccupant—Alignment`	The alignment.
`LStream*`	`inStream`	The stream to read from.

~LPlaceHolder()

Purpose The destructor destroys the object.

Access Virtual, Public

Prototype

```
virtual ~LPlaceHolder();
```

CountPanels()

Purpose Return the number of horizontal and vertical Panels. A Panel is a "frameful" of a View's Image. For an LPlaceHolder, the number of Panels is the number of Panels in its occupant Pane.

Access Virtual, Public

Prototype

```
virutal void CountPanels( Uint32 &outHorizPanels,
     Uint32 &outVertPanels);
```

Parameters This method has the following parameters:

`Uint32&`	`outHorizPanels`	The number of horizontal panels.
`Uint32&`	`outVertPanels`	The number of vertical panels.

Return None

InstallOccupant()

Purpose Install a pane inside an LPlaceHolder.

Use the Mac OS Toolbox `IconAlignmentType` values (in `<Icons.h>`) for `inAlignment`. If you don't specify a horizontal alignment, the Occupant width is set to the PlaceHolder width. Similarly, if you don't specify a vertical alignment, the Occupant height is set to the PlaceHolder height.

Access Public

Prototype

```
void InstallOccupant( LPane *inOccupant,
     Int16 inAlignment );
```

Parameters This method has the following parameter:

`LPane*`	`inOccupant`	The occupant.
`Int16`	`inAlignment`	Mac OS Toolbox `IconAlignmentType` values (in `<Icons.h>`). This is an optional parameter, if not specified, the default value (of -1) means to use the alignment stored in the PlaceHolder.

Return None

RemoveOccupant()

Purpose Remove occupant from a PlaceHolder, restoring the occupant to its original state.

Access Public

Prototype
```
LPane* RemoveOccupant();
```

Parameters None

Return The pointer to the occupant.

ScrollToPanel()

Purpose Scroll view image to the specified panel. For a PlaceHolder, scroll occupant pane to the specified panel.

Access Virtual, Public

Prototype
```
virutal Boolean ScrollToPanel(
     const PanelSpec &inPanel );
```

Parameters This method takes the following parameters:

`const PanelSpec&`	`inPanel`	The panel to scroll.

Return None

mOccupant

Purpose The occupant pointer.

Access Protected

Prototype
```
LPane *mOccupant;
```

mOccupantSuperView

Purpose The occupant superview pointer.

Access Protected

Prototype
```
LView *mOccupantSuperView;
```

mOccupantPlaceH

Purpose The occupant handle.

Access Protected

Prototype
```
Handle mOccupantPlaceH;
```

mOccupantAlignment

Purpose Occupant alignment parameters.

Access Protected

Prototype
```
Int16 mOccupantAlignment;
```

LPreferencesFile

Overview LPreferencesFile is a PowerPlant class that is used for creating preferences files for your application.

Methods The methods in this class are:

LPreferencesFile()

OpenOrCreateResourceFork()

Data Members There are no data members for this class.

Operation This is a very simple class.

Source files (File & StreamClasses)

`LPreferencesFile.h`

`LPreferencesFile.cp`

Ancestors LFile

LPreferencesFile()

Purpose The constructors create objects from the passed-in values.

Access Public

Prototype

```
LPreferencesFile();
LPreferencesFile( ConstStr255Param inFileName,
     Boolean inCreateFolder);
LPreferencesFile( FSSpec& inFileSpec );
LPreferencesFile( AliasHandle inAlias,
     Boolean& outWasChanged,
     FSSpec* inFromFile );
```

Parameters The parameters for these constructors are:

`ConstStr255Param`	`inFileName`	The name of the file to create.
`Boolean`	`inCreateFolder`	This is passed to `FindFolder()`.
`FSSpec&`	`inFileSpec`	The file spec for the file to use.
`AliasHandle`	`inAlias`	A handle to an alias to use.
`Boolean`	`outWasChanged`	Indicates if the `AliasHandle` was changed during resolution.
`FSSpec*`	`inFromFile`	A File Specifier for the starting point for a relative search. If nil, an absolute search is performed.

OpenOrCreateResourceFork()

Purpose .Opens the resource fork of the preferences file, if it exists. If the resource fork doesn't exist, it creates one. If the file doesn't exist, it creates the file with the designated type and creator codes.

Access Virtual, Public

Prototype

```
virtual Int16 OpenOrCreateResourceFork(
     Int16 inPrivileges,
     OSType inCreator,
     OSType inFileType,
     ScriptCode inScriptCode );
```

Parameters The parameters for these constructors are:

`Int16`	`inPrivileges`	The privileges to use.
`OSType`	`inCreator`	The creator type to use.
`OSType`	`inFileType`	The file type to use.
`ScriptCode`	`inScriptCode`	The script code to use.

Return Returns the `RefNum` of the resource fork for the file created.

LPrintout

Overview

LPrintout is a PowerPlant class that is used for support of printing in your application.

Methods

The methods in this class are:

LPrintout()	~LPrintout()
ApplyForeAndBackColors()	CountPanels()
CreatePrintout()	DoPrintJob()
EstablishPort()	FinishCreateSelf()
GetMacPort()	GetPrintJobSpecs()
GetPrintRecord()	HasAttribute()
InitPrintout()	PageToPanel()
PrintCopiesOfPages()	PrintPanel()
PrintPanelRange()	SetAttribute()
SetForeAndBackColors()	SetPrintRecord()

Data Members

The data members in this class are:

mAttributes	mPrintRecordH
mPrinterPort	mWindowPort
mHorizPanelCount	mVertPanelCount
mForeColor	mBackColor

Operation

To learn more about working with LPrintout, refer to *The PowerPlant Book*.

Source files

(Pane Classes)

`LPrintout.h`

`LPrintout.cp`

Ancestors LAttachable

LPane

LView

LPrintout()

Purpose The constructors create objects from the passed-in parameters.

Access Public

Prototype
```
LPrintout();
LPrintout( THPrint inPrintRecordH );
LPrintout( LStream *inStream );
```

Parameters The parameters for these constructors are:

`THPrint`	`inPrintRecordH`	The print record handle.
`LStream*`	`inStream`	The stream to read from.

~LPrintout()

Purpose The destructor destroys the object.

Access Virtual, Public

Prototype
```
virtual ~LPrintout();
```

ApplyForeAndBackColors()

Purpose Set the foreground and background colors of the current port. The Printout or one of its subpanes must already be focused. This is an override of ApplyForeAndBackColors() in LPane.

CountPanels()

Purpose Count the number of Panels in a Printout. This is an override of CountPanels() in LPane.

CreatePrintout()

Purpose Return a new Printout object (and its SubPanes) from the data in a `'PPob'` resource.

Access Static, Public

Prototype

```
static LPrintout* CreatePrintout(
     ResIDT inPrintoutID );
```

Parameters The parameter for this method is:

`ResIDT`	`inPrintoutID`	The resource ID for the printout.

Return The pointer to the new object.

DoPrintJob()

Purpose Print the job.

Access Virtual, Public

Prototype

```
virtual void DoPrintJob();
```

Parameters None

Return None

EstablishPort()

Purpose Make Printout the current port.

Access Virtual, Public

Prototype `virtual Boolean EstablishPort();`

Parameters None

Return Boolean indicating whether the port was successfully set, `false` if not.

FinishCreateSelf()

Purpose This is an override of FinishCreateSelf() in LPane.

GetMacPort()

Purpose Return the GrafPort associated with a Printout. This is an override of GetMacPort() in LPane.

GetPrintJobSpecs()

Purpose Extract information about the print job from the Mac OS Toolbox PrintRecord.

- First Page to print (mapped to a PanelSpec)
- Last Page to print (mapped to a PanelSpec)
- Number of copies to print

Access Virtual, Protected

Prototype

```
virtual void GetPrintJobSpecs(
     PanelSpec &outFirstPanel,
     PanelSpec &outLastPanel,
     Uint16 &outCopyCount );
```

Parameters The parameters for this method are:

`PanelSpec&`	`outFirstPanel`	The first panel.
`PanelSpec&`	`outLastPanel`	The last panel.
`Uint16&`	`outCopyCount`	The number of copies.

Return None

GetPrintRecord()

Purpose Retrieve the PrintRecord.

Access Public

Prototype
```
THPrint GetPrintRecord() const;
```

Parameters None

Return A handle to the PrintRecord.

Remarks None

HasAttribute()

Purpose Return whether a Printout has the specified attribute.

Access Public

Prototype
```
Boolean HasAttribute(
     EPrintAttr inAttribute ) const;
```

Parameters The parameter for this method is:

`EPrintAttr`	`inAttribute`	The attribute to check for.

Return Boolean indicating whether the attribute is present (`true`) or not (`false`).

InitPrintout()

Purpose Initialize data members to default values.

Access Private

Prototype `void InitPrintout();`

Parameters None

Return None

PageToPanel()

Purpose Fill in PanelSpec for a given page number.

Access Virtual, Protected

Prototype
```
virtual void PageToPanel(
     Uint32 inPageNumber,
     PanelSpec &outPanel ) const;
```

Parameters The parameters for this method are:

`Uint32`	`inPageNumber`	The page number to check on.
`PanelSpec&`	`outPanel`	The filled-in panelspec.

Return None

PrintCopiesOfPages()

Purpose Print copies of the specified range of Panels. This is the main "print loop."

Access Virtual, Protected

Prototype
```
virtual void PrintCopiesOfPages(
     const PanelSpec &inFirstPanel,
     const PanelSpec &inLastPanel,
     Uint16 inCopyCount );
```

Parameters The parameters for this method are:

`Uint16`	`inCopyCount`	The page number to check on.
`PanelSpec&`	`inFirstPanel`	The first panel.
`PanelSpec&`	`inLastPanel`	The last panel.

PrintPanel()

Purpose Print the specified Panel

Access Virtual, Public

Prototype

```
virtual void PrintPanel(
     const PanelSpec &inPanel,
     RgnHandle inSuperPrintRgnH );
```

Parameters The parameters for this method are:

`const PanelSpec&`	`inPanel`	The panel spec.
`RgnHandle`	`inSuperPrintRgnH`	The region handle.

PrintPanelRange()

Purpose Print the range of panels.

Access Virtual, Public

Prototype

```
virtual void PrintPanelRange(
     const PanelSpec &inFirstPanel,
     const PanelSpec &inLastPanel,
     Uint16 inCopyCount );
```

Parameters The parameters for this method are:

`const PanelSpec&`	`inFirstPanel`	The first panel.
`const PanelSpec&`	`inLastPanel`	The last panel.
`RgnHandle`	`inSuperPrintRgnH`	The region handle.

Return None

SetAttribute()

Purpose Specify attributes for a Printout.

Access Public

Prototype
```
void SetAttribute( EPrintAttr inAttribute );
```

Parameters The parameter for this method is:

`EPrintAttr`	`inAttribute`	The region handle.

Return None

SetForeAndBackColors()

Purpose Sprecify the foreground and/or background colors of a Printout. Specify `nil` for inForeColor and/or inBackColor to leave that color trait unchanged. This is an override of SetForeAndBackColors() in LPane.

SetPrintRecord()

Purpose Set the Toolbox PrintRecord for a Printout.

Access Public

Prototype
```
void SetPrintRecord(
     THPrint inPrintRecordH );
```

Parameters The parameter for this method is:

`THPrint`	`inPrintRecordH`	The handle to print record.

Return None

mAttributes

Purpose The attributes.

Access Protected

Prototype
```
Uint32 mAttributes;
```

mPrintRecordH

Purpose The handle PrintRecord.

Access Protected

Prototype
```
THPrint mPrintRecordH;
```

mPrinterPort

Purpose The printer port.

Access Protected

Prototype
```
TPPrPort mPrinterPort;
```

mWindowPort

Purpose The window pointer.

Access Protected

Prototype `WindowPtr mWindowPort;`

mHorizPanelCount

Purpose

Access Protected

Prototype `Uint32 mHorizPanelCount;`

mVertPanelCount

Purpose The vertical panel count.

Access Protected

Prototype `Uint32 mVertPanelCount;`

mForeColor

Purpose The foreground color.

Access Protected

Prototype `RGBColor mForeColor;`

mBackColor

Purpose The background color.

Access Protected

Prototype `RGBColor mBackColor;`

LQueue

Overview LQueue is a PowerPlant class that is used for implementing full linked list behavior for a list of LLink objects.

Methods The methods in this class are:

LQueue()	~LQueue()
DoForEach()	GetSize()
IsEmpty()	NextGet()
NextPut()	Remove()
operator =()	

Data Members The data members in this class are:

mFirst	mLast
mSize	

Operation Every element in LQueue is an LLink object. The methods also work on LLink objects. LQueue implements a first-in, first-out (FIFO) queue. You add elements to the end of the list, and you may only get items from the head of the list.

Source files (Threads Classes)

`LQueue.h`

`LQueue.cp`

See also LSharedQueue

LLink

LQueue()

Purpose The constructor creates the object.

Access Public

Prototype `LQueue();`

Parameters None

~LQueue()

Purpose The destructor destroys the LQueue object.

Access Virtual

Prototype
```
virtual ~LQueue();
```

DoForEach()

Purpose Execute a user-supplied function for each element in the queue.

Access Virtual, Public

Prototype
```
virtual void DoForEach(LQueueIterator proc,
     void* arg);
```

Parameters The parameters for this method are:

`LQueueIterator`	`proc`	The queue iterator.
`void*`	`arg`	The function to perform.

Return None

GetSize()

Purpose Returns the number of elements in the queue.

Access Virtual, Public

Prototype
```
virtual Uint32 GetSize() const;
```

Parameters None

Return The number of items in the queue.

IsEmpty()

Purpose Indicates if the queue not contain any elements.

Access Virtual, Public

Prototype
```
virtual Boolean IsEmpty() const;
```

Parameters None

Return Return true if the queue is empty.

NextGet()

Purpose Removes and returns the first element in the queue.

Access Virtual, Public

Prototype
```
virtual LLink* NextGet();
```

Parameters None

Return A LLink pointer to the element in the queue.

NextPut()

Purpose Adds the given element to the end of the queue.

Access Virtual, Public

Prototype
```
virtual void NextPut(LLink* inLinkP);
```

Parameters A pointer to an LLink

Return None

Remove()

Purpose Remove an arbitrary element from the queue. This function traverses the entire queue, looking for the given queue element. If the element is found, it is removed from the queue.

Access Virtual, Public

Prototype
```
virtual Boolean Remove(LLink *inLinkP);
```

Parameters A pointer to an LLink

Return Returns a Boolean indicating if the element was found.

operator =()

Purpose The assignment operator

Access Private

Prototype
```
LQueue&  operator = (const LQueue&);
```

Parameters An LQueue object to assign

Return Assigned object

mFirst

Purpose The head of the queue.

Access Protected

Prototype
```
LLink* mFirst;
```

mLast

Purpose The tail of the queue.

Access Protected

Prototype `LLink* mLast;`

mSize

Purpose The size of the queue in elements.

Access Protected

Prototype `Uint32 mSize;`

LRadioGroup

Overview

LRadioGroup is a PowerPlant class that is used for managing a group of Controls by ensuring that only one Control in a group is "on" at any time. This is the standard behavior of a set of Radio Buttons.

Methods

The methods in this class are:

LRadioGroup()
~LRadioGroup()
AddRadio()
GetCurrentRadioID()
ListenToMessage()

Data Members

The data members in this class are:

mCurrentRadio

Operation

Although you will normally use this class with StdRadioButton objects, the group members can be any kind of Control. Therefore, you can group your own custom Control objects. This class assumes the "off" value is zero, and the "on" value is one.

RadioGroup and its Controls have a Listener/Broadcaster relationship.

Source files

(Support Classes)

`LRadioGroup.h`

`LRadioGroup.cp`

See also

LControl

LListener

LRadioGroup()

Purpose

The constructors create objects from the passed-in parameters.

Access Public

Prototype
```
LRadioGroup();
LRadioGroup( LStream *inStream );
```

Parameters The stream constructor has the following parameter:

`LStream*`	`inStream`	A pointer to a stream object that contains the information to create the LRadioGroup object.

~LRadioGroup()

Purpose The destructor destroys the object.

Access Virtual, Public

Prototype
```
virtual ~LRadioGroup();
```

AddRadio()

Purpose Adds a Control to a Radio Group.

If the Control is "on" it becomes the current radio, turning off the former current radio.

Access Virtual, Public

Prototype
```
virtual void AddRadio( LControl *inRadio );
```

Parameters The parameter for this method is:

`LControl*`	`inRadio`	The pointer to the LControl object that we wish to add to.

Return None

GetCurrentRadioID()

Purpose Return the PaneID of the current RadioButton in the Group.

Access Virtual, Public

Prototype
```
virtual PaneIDT GetCurrentRadioID();
```

Parameters None

Return PaneIDT

ListenToMessage()

Purpose React to Messages from Broadcasters, which must be the Controls in the group. This is a pure virtual method (ListenToMessage()) in the base class which you must override. Refer to the description there for more information.

mCurrentRadio

Purpose This data member holds the pointer to the current radio button.

Access Protected

Prototype
```
LControl *mCurrentRadio;
```

LReentrantMemoryPool

Overview

LReentrantMemoryPool is a PowerPlant class that is areentrant memory manager for use by interrupt callbacks. Based on the `NEWMODE_FAST` implementation of operator `new`.

Methods

The methods in this class are:

LReentrantMemoryPool()	~LReentrantMemoryPool()
AddPool()	AllocFrom()
DisposePtr()	FreeMem()
GetPtrSize()	MakeFreeBlock()
NewPtr()	NewPtrClear()
TotalMem()	

Data Members

The data members in this class are:

mMemoryPools	mFreeBlocks;

Source files

(Networking Classes)

`LReentrantMemoryPool.h`

`LReentrantMemoryPool.cp`

See also

LSharedMemoryPool

LReentrantMemoryPool()

Purpose

The constructor creates the object, specifying the initial size of the memory pool.

IMPORTANT: MUST NOT BE CALLED AT INTERRUPT TIME!

Access

Public

Prototype

```
LReentrantMemoryPool (UInt32 inInitialSize);
LReentrantMemoryPool();// do not use
LReentrantMemoryPool(LReentrantMemoryPool&);
```

Parameters The parameters for this constructor are:

`UInt32`	`inInitialSize`	The initial size of the pool.

~LReentrantMemoryPool()

Purpose The destructor destroys the object.

Access Virtual, Public

Prototype

```
virtual ~LReentrantMemoryPool();
```

AddPool()

Purpose Add a memory pool.

Access Public

Prototype

```
void AddPool (UInt32 inPoolSize);
```

Parameters The parameters for this method are:

`UInt32`	`inPoolSize`	The size of the pool.

Return None

AllocFrom()

Purpose Allocate memory from the pool.

Access Private

Prototype

```
void* AllocFrom (SFreeBlockInfo* inBlockInfo,
```

```
UInt32 inByteCount);
```

Parameters The parameters for this method are:

`SFreeBlockInfo*`	`inBlockInfo`	The block information.
`UInt32`	`inByteCount`	The number of bytes.

Return The buffer pointer.

DisposePtr()

Purpose Free a non-relocatable block from this memory pool. May be called at interrupt time.

Access Public

Prototype

```
void DisposePtr (void* inPtr);
```

Parameters The parameters for this method are:

`void*`	`inPtr`	The pointer.

Return None

FreeMem()

Purpose Return the total amount of free memory in this memory pool. May be called at interrupt time.

Access Public

Prototype

```
UInt32 FreeMem (void);
```

Parameters None

Return The amount of free memory in the pool.

GetPtrSize()

Purpose Return the size of the non-relocatable block at `inPtr`. NOTE: The size will be rounded up to the nearest multiple of 4. May be called at interrupt time.

Access Public

Prototype

```
UInt32 GetPtrSize (void* inPtr);
```

Parameters The parameters for this method are:

`void*`	`inPtr`	The pointer.

Return The size of the memory block.

MakeFreeBlock()

Purpose Create a free block.

Access Private

Prototype

```
void MakeFreeBlock (void* inBlock,
     UInt32 inByteCount);
```

Parameters The parameters for this method are:

`void*`	`inBlock`	The pointer.
`UInt32`	`inByteCount`	The byte count.

Return None

NewPtr()

Purpose Allocate a new non-relocatable block within the reentrant memory pool. May be called at interrupt time.

Access Public

Prototype
```
void* NewPtr (UInt32 inByteCount);
```

Parameters The parameters for this method are:

`UInt32`	`inByteCount`	The byte count.

Return The pointer.

NewPtrClear()

Purpose Allocate a new non-relocatable block and clear it. May be called at interrupt time.

Access Public

Prototype
```
void* NewPtrClear (UInt32 inByteCount);
```

Parameters The parameters for this method are:

`UInt32`	`inByteCount`	The byte count.

Return The pointer.

TotalMem()

Purpose Return the total amount of memory used by this memory pool. May be called at interrupt time.

Access Public

Prototype
```
UInt32 TotalMem (void);
```

Parameters None

Return The total memory in the pool.

mMemoryPools

Purpose The pools.

Access Private

Prototype
```
LInterruptSafeList mMemoryPools;
```

mFreeBlocks;

Purpose The free blocks in the pool.

Access Private

Prototype
```
LInterruptSafeList mFreeBlocks;
```

LScroller

Overview

LScroller is a PowerPlant class that is used for scrolling views. Scroller views implement all the functionality of scroll bars, including creation, resizing, moving, hiding, showing, enabling and other operations. Sometimes you will see scrolling views referred to simply as "Scrollers" in the PowerPlant documentation.

The only functional difference between LActiveScroller and LScroller is that LActiveScroller supports dynamic scrolling of the view

Methods

The methods in this class are:

LScroller()	~LScroller()
ActivateSelf()	AdjustScrollBars()
DeactivateSelf()	DrawSelf()
ExpandSubPane()	FinishCreateSelf()
HasHorizontalScrollBar()	HasVerticalScrollBar()
HorizSBarAction()	HorizScroll()
InstallView()	ListenToMessage()
MakeScrollBars()	ResizeFrameBy()
SubImageChanged()	VertSBarAction()
VertScroll()	

Data Members

The data members in this class are:

mScrollingView	mVerticalBar
mHorizontalBar	mScrollingViewID

Operation

The most important feature of a Scroller view is that it has only one subpane, which is a view. The subview may contain an arbitrary number of panes and views. The net effect is that the Scroller view may contain any number of panes of any type.

Source files (Pane Classes)

`LScroller.h`

`LScroller.cp`

Ancestors LListener

LPane

LView

LScroller()

Purpose The constructor creates objects from the passed-in parameters.

Access Public

Prototype

```
LScroller();
LScroller( const LScroller &inOriginal );
LScroller( const SPaneInfo &inPaneInfo,
     const SViewInfo &inViewInfo,
     Int16 inHorizBarLeftIndent,
     Int16 inHorizBarRightIndent,
     Int16 inVertBarTopIndent,
     Int16 inVertBarBottomIndent,
     LView *inScrollingView );
LScroller( LStream *inStream );
```

Parameters These constructors have the following parameters:

`const LScroller&`	`inOriginal`	A reference to the LScroller object you want to copy.
`const SPaneInfo&`	`inPaneInfo`	A reference to the SPaneInfo object that is the super view.
`const SViewInfo&`	`inViewInfo`	A reference to the SViewInfo object that contains information about the SuperView.

`Int16`	`inHorizBarLeft-Indent`	The indentation to use on the left side for the horizontal scrolling. A good initial value for this parameter is 15.
`Int16`	`inHorizBar—RightIndent`	The indentation to use on the right side for the horizontal scrolling. A good initial value for this parameter is 15.
`Int16`	`inVertBarTop—Indent`	The indentation to use on the top for the vertical scrolling. A good initial value for this parameter is 15.
`Int16`	`inVertBarBot-tomIndent`	The indentation to use on the bottom for the vertical scrolling. A good initial value for this parameter is 15.
`LView`	`inScrollingView`	A pointer to the view that corresponds to this Scroller.
`LStream*`	`inStream`	A pointer to a stream object that contains the information to create the LScroller object.

~LScroller()

Purpose The destructor destroys the object.

Access Virtual, Public

Prototype

```
virtual ~LScroller();
```

ActivateSelf()

Purpose This method activates the Scroller, and show scroll bars that were hidden when deactivated. This method is an override of ActivateSelf() in the LPane class.

AdjustScrollBars()

Purpose This method adjusts the scroll bars (value, min, and max) according to the current state of the Scroller and ScrollingView.

Access Virtual, Public

Prototype `virtual void AdjustScrollBars();`

Parameters None

Return None

DeactivateSelf()

Purpose This method deactivates the Scroller. According to Mac OS Human Interface Guidelines, scroll bars in inactive windows are hidden. This method is an override of DeactivateSelf() in the LPane class.

DrawSelf()

Purpose This method draws a Scroller. This method is an override of DrawSelf() in the LPane class.

ExpandSubPane()

Purpose Expand a SubPane, which should be the Scroller's ScrollingView, to fill the interior of a Scroller. This method is an override of ExpandSubPane() in the LView class.

FinishCreateSelf()

Purpose Finish creation of a Scroller by installing its ScrollingView. This method is an override of FinishCreateSelf() in the LPane class.

HasHorizontalScrollBar()

Purpose This method returns a `Boolean` indicating whether mHorizontalBar is nil or not.

Access Inline, Public

Prototype `Boolean HasHorizontalScrollBar();`

Parameters None

Return `Boolean` indicating mHorizontalBar != `nil`

HasVerticalScrollBar()

Purpose This method returns a `Boolean` indicating whether mVerticalBar is nil or not.

Access Inline, Public

Prototype `Boolean HasVerticalScrollBar();`

Parameters None

Return `Boolean` indicating mVerticalBar != `nil`

HorizSBarAction()

Purpose Mac OS Toolbox callback function for the action to take while tracking a mouse click in a horizontal scroll bar.

Access Static, Protected

Prototype

```
static pascal void HorizSBarAction(
     ControlHandle inMacControl,
     Int16 inPart);
```

Parameters This method has the following parameters:

`ControlHandle`	`inMacControl`	This is the handle to the Mac OS control.
`Int16`	`inPart`	This is an enumeration, like `kControl—UpButtonPart`. See the Mac OS system header file named Controls.h for a list of these.

Return None

HorizScroll()

Purpose This method is called to scroll horizontally while clicking and holding inside the horizontal scroll bar.

Access Virtual, Public

Prototype

```
virtual void HorizScroll( Int16 inPart );
```

Parameters This method has the following parameter:

`Int16`	`inPart`	This is an enumeration, like `kControl—UpButtonPart`. See the Mac OS system header file named Controls.h for a list of these.

Return None

InstallView()

Purpose This method installs a Scrolling View within this Scroller.

Access Virtual, Public

Prototype

```
virtual void InstallView( LView *inScrollingView );
```

Parameters This method has the following parameter:

`LView*`	`inScrollingView`	This is a pointer to the scrolling view to install in the scroller.

Return None

ListenToMessage()

Purpose This method responds to messages from Broadcasters. The ScrollBars of a Scroller broadcast a message after the user drags the thumb. This method is an override of the pure virtual method ListenToMessage() in the LListener class.

Access Virtual, Public

Prototype

```
virtual void ListenToMessage( MessageT inMessage,
     void *ioParam);
```

Parameters This method has the following parameters:

`MessageT`	`inMessage`	This is the message that is broadcast from the framework. If the message is `msg_ThumbDragged` then processing is required.
`void*`	`ioParam`	This is a data block pointer that accompanies the message.

Return None

Remarks You must supply an implementation for this method since it is a pure virtual in the base class.

MakeScrollBars()

Purpose Create standard Mac OS control objects for the horizontal and/or vertical scroll bars of a Scroller.

Access Private

Prototype

```
void MakeScrollBars( Int16 inHorizBarLeftIndent,
     Int16 inHorizBarRightIndent,
     Int16 inVertBarTopIndent,
     Int16 inVertBarBottomIndent );
```

Parameters This method has the following parameters:

`Int16`	`inHorizBar-LeftIndent`	The indentation to use on the left side for the horizontal scrolling. A good initial value for this parameter is 15.
`Int16`	`inHorizBar-RightIndent`	The indentation to use on the right side for the horizontal scrolling. A good initial value for this parameter is 15.
`Int16`	`inVertBar-TopIndent`	The indentation to use on the top for the vertical scrolling. A good initial value for this parameter is 15.
`Int16`	`inVertBar-BottomIndent`	The indentation to use on the bottom for the vertical scrolling. A good initial value for this parameter is 15.

Return None

ResizeFrameBy()

Purpose Change the Frame size by the specified amounts. This method is an override of ResizeFrameBy() in the LPane class.

SubImageChanged()

Purpose Adjust state when the Image of the ScrollingView changes. Scroll bar settings depend on the ScrollingView Image, so they must be adjusted to match the current state. This method is an override of SubImageChanged() in the LView class.

Access Virtual, Public

Prototype

```
virtual void SubImageChanged( LView *inSubView );
```

Parameters This method has the following parameter:

`LView*`	`inSubView`	This is a pointer to the scrolling view to install in the scroller.

Return None

VertSBarAction()

Purpose Mac OS Toolbox callback function for the action to take while tracking a mouse click in a vertical scroll bar.

Access Static, Protected

Prototype

```
static pascal void VertSBarAction(
     ControlHandle inMacControl,
     Int16 inPart);
```

Parameters This method has the following parameters:

`ControlHandle`	`inMacControl`	This is the handle to the Mac OS control.
`Int16`	`inPart`	This is an enumeration, like `kControlUpBut-tonPart`. See the Mac OS system header file named Controls.h for a list of these.

Return None

VertScroll()

Purpose This method is called to scroll vertically while clicking and holding inside the vertical scroll bar.

Access Virtual, Public

Prototype

```
virtual void VertScroll( Int16 inPart );
```

Parameters This method has the following parameter:

`Int16`	`inPart`	This is an enumeration, like `kContro-lUpButtonPart`. See the Mac OS system header file named Controls.h for a list of these.

Return None

mScrollingView

Purpose This data member is a pointer to the ScrollingView.

Access Protected

Prototype `LView *mScrollingView;`

mVerticalBar

Purpose This data member is a pointer to the standard control that is a vertical scroll bar.

Access Protected

Prototype `LStdControl *mVerticalBar;`

mHorizontalBar

Purpose This data member is a pointer to the standard control that is a horizontal scroll bar.

Access Protected

Prototype `LStdControl *mHorizontalBar;`

mScrollingViewID

Purpose This data member is the resource ID of the ScrollingView.

Access Protected

Prototype `PaneIDT mScrollingViewID;`

LSemaphore

Overview

LSemaphore is a PowerPlant class that is used for implementing semaphores. Most of the methods and members are internal to PowerPlant.

Methods

The methods in this class are:

LSemaphore()	~LSemaphore()
BlockThread()	InitSemaphore()
Signal()	UnblockAll()
UnblockThread()	Wait()
operator =()	

Data Members

The data members in this class are:

mExcessSignals0	mThreads

Operation

The only two methods you will typically concern yourself with are Wait() and Signal().

Source files

(Threads Classes)

`LSemaphore.h`

`LSemaphore.cp`

See also

LEventSemaphore

LMutexSemaphore

LThread

LSemaphore()

Purpose

The constructor creates the object.

Access

Public

Prototype

```
LSemaphore();
LSemaphore(Int32 initialCount);
```

Copy Constructor

```
LSemaphore(const LSemaphore&);
```

Parameters Int32 containing the initial count to set the semaphore at, or a semaphore to make a copy of.

~LSemaphore()

Purpose The destructor destroys the LSemaphore object.

Access Virtual, Public

Prototype

```
~LSemaphore();
```

BlockThread()

Purpose Block the current thread for the given number of milliseconds, refer to Wait().

NOTE:The current thread must be in a 1-level-deep critical section upon entering this function!

Access Protected

Prototype

```
ExceptionCode BlockThread(Int32 milliSeconds);
```

Parameters Int32 containing the number of msec to block for.

Return An exception code

InitSemaphore()

Purpose Initializes the fields of a semaphore.

Access Private

Prototype `void InitSemaphore(Int32 initialCount);`

Parameters The initial count to set the semaphore to.

Return None

Signal()

Purpose Release a semaphore. If any threads are waiting on the semaphore, one of them is unblocked.

Access Virtual, Public

Prototype `virtual void Signal();`

Parameters None

Return None

UnblockAll()

Purpose Unblock all of the threads waiting on this semaphore, passing them the given error code.

Access Protected

Prototype `void UnblockAll(ExceptionCode error);`

Parameters The exception code

Return None

UnblockThread()

Purpose Unblock the thread pointed to by qEl, passing it the given error code.

NOTE:The current thread must be in a critical section upon entering this method!

Access Protected

Prototype

```
LThread* UnblockThread(QElemPtr qEl,
   ExceptionCode error);
```

Parameters The parameters for this method are:

`QElemPtr`	`qEl`	The queue element.
`ExceptionCode`	`error`	The exception code.

Return A pointer to the thread

Wait()

Purpose Wait for a semaphore to become available. An optional argument specifies how long the caller is willing to wait for the semaphore to become available. Possible values are:

- `semaphore_WaitForever`Wait indefinitely (default).
- `semaphore_NoWait`Do not wait. If the semaphore is unavailable, return `errSemaphoreTimedOut`.
- other (> 0)Wait time in milliseconds

If the time interval expires before the semaphore becomes available, this function returns errSemaphoreTimedOut.

Access Virtual, Public

Prototype

```
ExceptionCode Wait(Int32 milliSeconds);
```

Parameters The time to wait in msec.

Return An exception code.

operator =()

Purpose The assignment operator

Access Private

Prototype
```
LSemaphore&operator = (const LSemaphore&);
```

Parameters A reference to a semaphore

Return A reference to a semaphore

mExcessSignals0

Purpose Excess signal / thread count

Access Protected

Prototype
```
Int32 mExcessSignals;
```

mThreads

Purpose List of waiting threads

Access Protected

Prototype
```
QHdr mThreads;
```

LSendQueue

Overview

LSendQueue is a PowerPlant class that queues data for asynchronous style data sending for MacTCP.

Methods

The methods in this class are:

LSendQueue()	~LSendQueue()
Append()	GetBlockingDataSize()
GetMaxPendingRelease()	Int_InternalSend()
Int_SendComplete()	InternalClearReleaseQueue()
IsBusy()	KillQueue()
NotifyRelease()	Run()
SetBlockingDataSize()	SetMaxPendingRelease()
WaitingDataSize()	

Data Members

The data members in this class are:

mWaitingQueue	mPendingQueue
mReleaseQueue	mContinue
mDeleteData	mEndpointDead
mBusy	mReleaseWaiting
mEndpoint	mThread
mMaxPendingRelease	mBlockingDataSize

Operation

This class is used primarily for Open Transport-style “acked” data transfers which are not native to MacTCP.

Source files

(Networking Classes)

`LSendQueue.h`

`LSendQueue.cp`

See also LInterruptSafeListMember

LMacTCPTCPSendQueue

LMacTCPUDPSendQueue

LThread

LSendQueue()

Purpose The constructor creates the object. This constructor inherits from LThread's constructor.

Access Public

Prototype

```
LSendQueue(LEndpoint * inEndpoint);
```

Parameters This constructor has the following parameter:

`LEndpoint*`	`inEndpoint`	This is the endpoint.

~LSendQueue()

Purpose The destructor destroys the object.

Access Virtual, Public

Prototype

```
virtual ~LSendQueue();
```

Append()

Purpose Perform an append on the queue.

Access Virtual, Public

Prototype

```
virtual void Append(
     LInterruptSafeListMember* inItem);
```

Parameters This method has the following parameter:

`LInterruptSafeListMember* inItemNumber`	This is the end-point.

Return None

GetBlockingDataSize()

Purpose This method returns the value of the mBlockingDataSize data member.

Access Virtual, Public

Prototype
```
virtual Uint32 GetBlockingDataSize() const;
```

Parameters None

Return The size of the blocking data.

GetMaxPendingRelease()

Purpose This method returns the value of the mMaxPendingRelease data member.

Access Virtual, Public

Prototype
```
virtual Uint32 GetMaxPendingRelease() const;
```

Parameters None

Return The value of mMaxPendingRelease.

Int_InternalSend()

Purpose This method is pure virtual, and you must supply an implementation for it.

Access Pure virtual, Protected

Prototype `virtual void Int_InternalSend() = 0;`

Parameters None

Return None

Int_SendComplete()

Purpose This routine *might* be called at interrupt time.

Access Virtual, Protected

Prototype `virtual void Int_SendComplete();`

Parameters None

Return None

InternalClearReleaseQueue()

Purpose Clear the release queue.

Access virtual Virtual, Protected

Prototype `void InternalClearReleaseQueue();`

Parameters None

Return None

IsBusy()

Purpose Returns the value of the mBusy data member.

Access Virtual, Public

Prototype `virtual Boolean IsBusy();`

Parameters None

Return The value of mBusy.

KillQueue()

Purpose Clear the queue and perform other necessary cleanup.

Access Virtual, Public

Prototype `virtual void KillQueue();`

Parameters None

Return None

NotifyRelease()

Purpose This is a pure virtual method that you must provide code for.

Access Pure virtual, Protected

Prototype `virtual void NotifyRelease(LSendData* inData) = 0;`

Parameters This method has the following parameter:

`LSendData*`	`inData`	This is the data to send.

Return None

Run()

Purpose Run only handles releasing data and the ACK'ing notifications. The data is sent via a chained notifier routine (started via Append()).

Access Virtual, Public

Prototype `virtual void* Run();`

Parameters None

Return Always returns nil.

SetBlockingDataSize()

Purpose Sets the value of mBlockingDataSize.

Access Virtual, Public

Prototype
```
virtual void SetBlockingDataSize(
     Uint32 inReleaseSize);
```

Parameters This method has the following parameter:

`Uint32`	`inReleaseSize`	The data buffer size.

Return None

SetMaxPendingRelease()

Purpose Sets the value of mMaxPendingRelease.

Access Virtual, Public

Prototype
```
virtual void SetMaxPendingRelease(
     Uint32 inReleaseSize);
```

Parameters This method has the following parameter:

`Uint32`	`inReleaseSize`	The data buffer size.

Return None

WaitingDataSize()

Purpose Returns the size of the pending data in the queue.

Access Virtual, Public

Prototype
```
virtual Uint32 WaitingDataSize();
```

Parameters None

Return The data size.

mWaitingQueue

Purpose The waiting queue.

Access Protected

Prototype
```
LInterruptSafeList mWaitingQueue;
```

mPendingQueue

Purpose The pending queue.

Access Protected

Prototype
```
LInterruptSafeList mPendingQueue;
```

mReleaseQueue

Purpose The release queue.

Access Protected

Prototype
```
LInterruptSafeList mReleaseQueue;
```

mContinue

Purpose Whether to contine.

Access Protected

Prototype `Boolean mContinue;`

mDeleteData

Purpose Whether to delete the data or not.

Access Protected

Prototype `Boolean mDeleteData;`

mEndpointDead

Purpose Whether the endpoint is dead or not.

Access Protected

Prototype `Boolean mEndpointDead;`

mBusy

Purpose Whether busy or not.

Access Protected

Prototype `Boolean mBusy;`

mReleaseWaiting

Purpose Whether a release is waiting.

Access Protected

Prototype `Uint32 mReleaseWaiting;`

mEndpoint

Purpose The endpoint.

Access Protected

Prototype
```
LEndpoint * mEndpoint;
```

mThread

Purpose The thread.

Access Protected

Prototype
```
LThread * mThread;
```

mMaxPendingRelease

Purpose The size of the maximum pending release buffer.

Access Protected

Prototype
```
Uint32 mMaxPendingRelease;
```

mBlockingDataSize

Purpose The size of the blocking data buffer.

Access Protected

Prototype
```
Uint32 mBlockingDataSize;
```

LSharable

Overview LSharable is a mix-in PowerPlant class that is used for reference counted objects.

Methods The methods in this class are:

LSharable() ~LSharable()

AddUser() GetUseCount()

NoMoreUsers() RemoveUser()

Data Members The data members in this class are:

mUseCount

Source files (Feature Classes)

`LSharable.h`

`LSharable.cp`

Ancestors LModelObject

LSharable()

Purpose The constructor creates objects.

Access Public

Prototype

```
LSharable();
```

~LSharable()

Purpose The destructor destroys the object.

Access Virtual, Protected

Prototype

```
virtual ~LSharable();
```

AddUser()

Purpose Add a user to an object.

Access Virtual, Public

Prototype
```
virtual void AddUser( void* inUser);
```

Parameters A pointer to the user.

Return None

GetUseCount()

Purpose Retrieve the use count of the object.

Access Virtual, Public

Prototype
```
virtual Int32 GetUseCount();
```

Parameters None

Return The number of users.

NoMoreUsers()

Purpose Internal function called when use count drops to zero.

Access Virtual, Protected

Prototype
```
virtual void NoMoreUsers();
```

Parameters None

Return None

RemoveUser()

Purpose Specify that a user is no longer sharing a Sharable object. This method ignores the user and just decrements the use count. Override if you wish to keep track users.

NOTE: Do not throw an exception from this method.

Access Virtual, Public

Prototype

```
virtual void RemoveUser (void* inUser);
```

Parameters A pointer to the user.

Return None

mUseCount

Purpose The user count.

Access Protected

Prototype

```
Int32mUseCount;
```

LSharableModel

Overview LSharableModel is a mix-in PowerPlant class that is used for reference counting LModelObjects.

Methods The methods in this class are:

LSharableModel() ~LSharableModel()

AddUser() Finalize()

SuperDeleted()

Data Members There are no data members in this class.

Source files (AppleEvent Classes)

`LSharableModel.h`

`LSharableModel.cp`

Ancestors LModelObject

LSharable

LSharableModel()

Purpose The constructor creates objects from the passed-in parameters.

Access Public

Prototype

```
LSharableModel();
LSharableModel( LModelObject*inSuperModel,
     DescTypeinKind = typeNull );
```

Parameters The parameters for these constructors are:

`LModelObject*`	`inSuperModel`	A pointer to the Super object.
`DescType`	`inKind`	The kind, which defaults to `typeNull`.

~LSharableModel()

Purpose The destructor destroys the object.

Access Virtual, Public

Prototype

```
virtual ~LSharableModel();
```

AddUser()

Purpose Add a user to the object. This method calls into the base class method AddUser() in LSharable also.

Access Virtual, Public

Prototype

```
virtual void AddUser(void *inUser);
```

Parameters The parameter for this method is:

`void*`	`inUser`	A pointer to the user object.

Return None

Finalize()

Purpose This calls the `Finalize()` method in the base class.

Access Virtual, Public

Prototype

```
virtual void Finalize();
```

Parameters None

Return None

SuperDeleted()

Purpose This is called when the Super Object is deleted.

Access Virtual, Public

Prototype

```
virtual void SuperDeleted(void);
```

Parameters None

Return None

LSharedMemoryPool

Overview LSharedMemoryPool is a PowerPlant class that implements static shared memory pools that are self-managing and create/delete as needed.

Methods The methods in this class are:

LSharedMemoryPool() ~LSharedMemoryPool()

AddPoolUser() GetSharedPool()

RemovePoolUser()

Data Members The data members in this class are:

sSharedMemoryPool

Operation A .

Source files (Networking Classes)

LSharedMemoryPool.h

LSharedMemoryPool.cp

See also LReentrantMemoryPool

LSharable

LSharedMemoryPool()

Purpose The constructor creates the object. This constructor inherits from the LReentrantMemoryPool constructor.

Access Public

Prototype

```
LSharedMemoryPool(Uint32 inPoolSize);
```

Parameters This constructor has the following parameter:

`Uint32`	`inPoolSize`	This is the size of the pool.

~LSharedMemoryPool()

Purpose The destructor destroys the object.

Access Virtual, Public

Prototype
```
virtual ~LSharedMemoryPool();
```

AddPoolUser()

Purpose Adds a user to the pool. Can NOT be called at interrupt time.

Access Public

Prototype
```
void AddPoolUser( void*inUser,
    Uint32inPoolSize);
```

Parameters This method has the following parameters:

`void*`	`inUser`	The user pointer.
`Uint32`	`inPoolSize`	This is the size of the pool.

Return None

GetSharedPool()

Purpose Returns pointer to static shared memory pool. Creates pool if necessary.

Access Static, Public

Prototype
```
static LSharedMemoryPool * GetSharedPool();
```

Parameters None

Return The pointer to the shared memory pool.

RemovePoolUser()

Purpose Removes a user from the pool.

Access Public

Prototype `void RemovePoolUser(void*inUser);`

Parameters This method has the following parameters:

`void*`	`inUser`	The user pointer.

Return None

sSharedMemoryPool

Purpose The shared memory poolk pointer.

Access Static, Private

Prototype `LSharedMemoryPool *sSharedMemoryPool;`

LSimpleThread

Overview LSimpleThread is a concrete subclass of LThread.

Methods The methods in this class are:

LSimpleThread() Run()

Data Members The data members in this class are:

mProc mArg

Operation When you instantiate an LSimpleThread object, you specify the ThreadProc pointer and the void pointer to data.

Source files (Threads Classes)

LSimpleThread.h

LSimpleThread.cp

See also LThread

LSimpleThread()

Purpose The constructor creates the object.

Access Public

Prototype

```
LSimpleThread(
     ThreadProc proc,
     void   *arg,
     Boolean  preempt,
     Uint32  stack,
     LThread::EThreadOption flags);
```

Parameters The parameters for this method are:

ThreadProc	proc	The thread proc.
void*	arg	The data arg.

`Boolean`	`preempt`	Preempt or not.
`Uint32`	`stack`	The stack size to use.
`LThread::EThreadOption`	`flags`	Thread option flags.

Run()

Purpose Start the thread.

Access Virtual, Protected

Prototype
```
virtual void *Run();
```

Parameters None

Return A pointer to the mResult

mProc

Purpose The procedure pointer

Access Private

Prototype
```
ThreadProc mProc;
```

mArg

Purpose The data argument

Access Private

Prototype
```
void* mArg;
```

LSingleDoc

Overview

LSingleDoc is a PowerPlant class that is used as a simple extension of LDocument. You must derive your own methods from this object ot provide functionality.

Methods

The methods in this class are:

LSingleDoc()	~LSingleDoc()
AllowSubRemoval()	GetDescriptor()
GetFile()	GetWindow()
MakeCurrent()	UsesFileSpec()

Data Members

The data members in this class are:

mWindow	mFile

Source files

(Commander Classes)

`LSingleDoc.h`

`LSingleDoc.cp`

See also

LCommander

LDocument

LSingleDoc()

Purpose

The constructor creates the object.

Access

Public

Prototype

```
LSingleDoc();
LSingleDoc( LCommander*inSuper );
```

Parameters

An LCommander super object.

~LSingleDoc()

Purpose The destructor destroys the object.

Access Virtual, Public

Prototype
```
virtual ~LSingleDoc();
```

AllowSubRemoval()

Purpose This is an override of AllowSubRemoval() in LCommander.

GetDescriptor()

Purpose This is an override of GetDescriptor() in LDocument.

GetFile()

Purpose This returns the file pointer held in mFile.

Access Public

Prototype
```
LFile*GetFile() const;
```

Parameters None

Return The file pointer.

GetWindow()

Purpose This method returns the window pointer held in mWindow.

Access Public

Prototype
```
LWindow*GetWindow() const;
```

Parameters None

Return None

MakeCurrent()

Purpose Make this Document the current one by selecting its Window.

Access Virtual, Public

Prototype
```
virtual void MakeCurrent();
```

Parameters None

Return None

UsesFileSpec()

Purpose Returns whether the Document's File has the given FSSpec.

Access Virtual, Public

Prototype
```
virtual Boolean UsesFileSpec(
     const FSSpec&inFileSpec ) const;
```

Parameters A FSSpec for the file.

Return Return true if it has the FSSpec.

mWindow

Purpose The window pointer.

Access Protected

Prototype
```
LWindow*mWindow;
```

mFile

Purpose The file pointer storage.

Access Protected

Prototype

```
LFile*mFile;
```

LSIOUXAttachment

Overview LSIOUXAttachment is a PowerPlant class that is used for using the SIOUX facilities within PowerPlant programs.

Methods The methods in this class are:

LSIOUXAttachment() ExecuteSelf()

Data Members There are no data members in this class.

Operation To learn more about SIOUX and how to use it, refer to the *MSL C Reference* on the CodeWarrior documentation CD.

Source files (Support Classes)

`LSIOUXAttachment.h`

`LSIOUXAttachment.cp`

See also LAttachment

LSIOUXAttachment()

Purpose The constructor configures SIOUX to run within a PowerPlant program.

Access Public

Prototype

```
LSIOUXAttachment();
```

Parameters None

Remarks If you want to change the default SIOUX values, set the fields of the global `SIOUXSettings` struct in your code before creating the SIOUX Window (normally the before the first call to `printf` or `cout`).

ExecuteSelf()

Purpose Send the Event to SIOUX. If SIOUX doesn't handle the event, `SIOUXHandleOneEvent`() returns 0, so `mExecuteHost` is true, meaning that the event will be dispatched to PowerPlant. If SIOUX does handle the event, `SIOUXHandleOneEvent`() returns a non-zero value, so `mExecuteHost` is false, meaning that the event will not be passed to PowerPlant.

Access Protected

Prototype

```
virtual void ExecuteSelf( MessageT inMessage,
     void *ioParam);
```

Parameters The parameters for this method are:

`MessageT`	`inMessage`	This parameter is currently unused for this method.
`void*`	`ioParam`	A pointer to a structure that contains `Rect` information for drawing.

Return None

LStdButton

Overview LStdButton is a PowerPlant class that is used for managing Mac OS push buttons.

Methods The methods in this class are:

LStdButton() HotSpotResult()

Data Members There are no data members in this class.

Operation Using this class, you specify the text to display in the button, the text traits resource ID to use for the text display, and the message to send when the button is clicked.

Source files (Pane Classes)

`LStdControl.h`

`LStdControl.cp`

Ancestors The ancestors this class derives from are:

LAttachable LControl LStdControl

LBroadcaster LPane

LStdButton()

Purpose The constructors create objects from the passed-in parameters.

Access Public

Prototype

```
LStdButton();
LStdButton( const LStdButton &inOriginal );
LStdButton( const SPaneInfo &inPaneInfo,
     MessageT inValueMessage,
     ResIDT inTextTraitsID,
     Str255 inTitle );
```

```
LStdButton( const SPaneInfo &inPaneInfo,
     MessageT inValueMessage,
     ResIDT inTextTraitsID,
     ControlHandle inMacControlH );
LStdButton( LStream *inStream );
```

Parameters These constructors have the following parameters:

`const LStdButton&`	`inOriginal`	A reference to the `LStdButton` object you want to copy.
`const SPaneInfo&`	`inPaneInfo`	A reference to the SPaneInfo object that is the super view.
`MessageT`	`inValueMessage`	The message sent when the button is depressed.
`ResIDT`	`inTextTraitsID`	The text traits resource ID for the button text.
`Str255`	`inTitle`	The text string to put on the button.
`ControlHandle`	`inMacControlH`	The handle for the Mac OS control.
`LStream*`	`inStream`	A pointer to a stream object that contains the information to create the LStdButton object.

HotSpotResult()

Purpose This method responds to a click in the button by broadcasting the value of the button. This method is an override of HotSpotResult() in the LControl class.

LStdCheckBox

Overview LStdCheckBox is a PowerPlant class that is used for managing Mac OS check box controls.

Methods The methods in this class are:

LStdCheckBox() HotSpotResult()

Data Members There are no data members in this class.

Operation If the user clicks the control. the value toggles between one and zero.

Source files (Pane Classes)

`LStdControl.h`

`LStdControl.cp`

Ancestors The ancestors this class derives from are:

LAttachable LControl LStdControl

LBroadcaster LPane

LStdCheckBox()

Purpose The constructors create objects from the passed-in parameters.

Access Public

Prototype

```
LStdCheckBox();
LStdCheckBox( const LStdCheckBox &inOriginal );
LStdCheckBox( const SPaneInfo &inPaneInfo,
     MessageT inValueMessage,
     Int32 inValue,
     ResIDT inTextTraitsID,
     Str255 inTitle );
```

```
LStdCheckBox( const SPaneInfo &inPaneInfo,
     MessageT inValueMessage,
     Int32 inValue,
     ResIDT inTextTraitsID,
     ControlHandle inMacControlH );
LStdCheckBox( LStream *inStream );
```

Parameters These constructors have the following parameters:

`const LStdCheckBox&`	`inOriginal`	A reference to the `LStdCheckBox` object you want to copy.
`const SPaneInfo&`	`inPaneInfo`	A reference to the SPaneInfo object that is the super view.
`MessageT`	`inValueMessage`	The message sent when the check box is clicked.
`Int32`	`inValue`	
`ResIDT`	`inTextTraitsID`	The text traits resource ID for the button text.
`Str255`	`inTitle`	The text string to put on the button.
`ControlHandle`	`inMacControlH`	The handle for the Mac OS control.
`LStream*`	`inStream`	A pointer to a stream object that contains the information to create the `LStdCheck-Box` object.

HotSpotResult()

Purpose This method responds to a click in the check box by toggling the value of the control between checked (value = 0) and unchecked (value = 1). This method is an override of HotSpotResult() in the LControl class.

LStdControl

Overview

LStdControl is a PowerPlant class that encapsulates the standard Mac OS control behavior.

Methods

The methods in this class are:

LStdControl()
~LStdControl()
AlignControlRect()
CalcBigValue()
CalcSmallValue()
CreateFromCNTL()
DisableSelf()
DrawSelf()
EnableSelf()
FindHotSpot()
FocusDraw()
GetDescriptor()
GetMacControl()
GetTrackingControl()
HideSelf()
HotSpotAction()
HotSpotResult()
InitStdControl()
MoveBy()
PointInHotSpot()
ResizeFrameBy()
SetActionProc()
SetDescriptor()
SetMaxValue()
SetMinValue()
SetStdMinAndMax()
SetThumbFunc()
SetValue()
ShowSelf()
TrackHotSpot()
ValueIsInStdRange()

Data Members

The data members in this class are:

mMacControlH
mThumbFunc
mControlKind
mTextTraitsID
mUsingBigValues
sTrackingControl

Operation

The specific design of this class supports scroll bar controls and controls that use custom control definitions (CDEFs). This class also forms the basis for the standard button, check box, pop-up menu, and radio button classes.

For detailed information on Mac OS controls, refer to *Inside Mac: MacintoshToolbox Essentials*, published by Addison-Wesley.

Source files

(Pane Classes)

`LStdControl.h`

`LStdControl.cp`

Ancestors

The ancestors for this class are:

LAttachable LControl

LBroadcaster LPane

LStdControl()

Purpose

The constructors create objects from the passed-in parameters. Many of these constructors call LControl base class constructors to force a portion of the initialization chores.

Access

Public

Prototype

```
LStdControl();
LStdControl( Int16 inControlKind );
LStdControl( const SPaneInfo &inPaneInfo,
     MessageT inValueMessage,
     Int32 inValue,
     Int32 inMinValue,
     Int32 inMaxValue,
     Int16 inControlKind,
     ResIDT inTextTraitsID,
     ConstStringPtr inTitle,
     Int32 inMacRefCon );
LStdControl( const SPaneInfo &inPaneInfo,
     MessageT inValueMessage,
```

```
            Int32 inValue,
            Int32 inMinValue,
            Int32 inMaxValue,
            Int16 inControlKind,
            ResIDT inTextTraitsID,
            ControlHandle inMacControlH );
LStdControl( LStream *inStream );
LStdControl( LStream *inStream, bool inDummy );
```

Parameters These constructors have the following parameters:

`Int16`	`inControlKind`	The value to set for mControlKind.
`const SPaneInfo&`	`inPaneInfo`	A reference to the SPaneInfo object that is the super view.
`MessageT`	`inValueMessage`	The message sent when the control is interacted with.
`Int32`	`inValue`	The value to set for the control.
`Int32`	`inMinValue`	The minimum value to allow for the control.
`Int32`	`inMaxValue`	The maximum value to allow for the control.
`ResIDT`	`inTextTraitsID`	The resource ID that contains the text traits for control text.
`ConstStringPtr`	`inTitle`	The title to use for the control.
`Int32`	`inMacRefCon`	The value to put in the Mac OS refcon storage space.
`ControlHandle`	`inMacControlH`	The Mac OS handle to the control.
`LStream*`	`inStream`	A pointer to a stream object that contains the information to create the LControl object.
`bool`	`inDummy`	This parameter is unused for this constructor.

Remarks

- `LStdControl( Int16 inControlKind )`—Construct a StdControl for a particular kind of Mac OS Toolbox Control. On entry, the current Port must be the Window into which to install the Control.

~LStdControl()

Purpose The destructor destroys the object.

Access Virtual, Public

Prototype

```
virtual ~LStdControl();
```

AlignControlRect()

Purpose Set the rectangle inside the Mac OS Toolbox `ControlRecord` to the Frame rectangle of the standard control.

Access Protected

Prototype

```
void AlignControlRect();
```

Parameters None

Return None

Remarks

- At (almost) all times, the contrlRect field of the Mac OS Toolbox ControlRecord must be the same as the Frame rectangle of the StdControl object. We store directly to the ControlRecord rather than using the SizeControl and MoveControl traps because we want to bypass the automatic drawing (and the need to set the port and coordinate system) performed by those traps.
- The contrlRect is not the same as the Frame rectangle if the latter is outside the 16-bit QuickDraw coordinate space.

CalcBigValue()

Purpose Map from Small to Big value when using 32 bit values. Call this function only when mUsingBigValues is true.

Access Protected

Prototype

```
Int32 CalcBigValue( Int16 inSmallValue );
```

Parameters This method has the following parameter:

`Int16`	`inSmallValue`	The Small value to be converted.

Return `Int32` containing the Big value.

Remarks The control class uses Big (32 bit) values, but the Mac Control Manager only supports Small (16 bit) values. See the discussion for the SetStdMinAndMax() method.

CalcSmallValue()

Purpose Map from Big to Small value when using 32 bit values. Call this function only when mUsingBigValues is true.

Access Protected

Prototype

```
Int16 CalcSmallValue( Int32 inBigValue );
```

Parameters This method has the following parameter:

`Int16`	`inBigValue`	The Big value to be converted.

Return `Int16` containing the Small value.

Remarks The control class uses Big (32 bit) values, but the Mac Control Manager only supports Small (16 bit) values. Refer to the discussion for the SetStdMinAndMax() method.

CreateFromCNTL()

Purpose Create a StdControl from a `'CNTL'` resource.

Access Static, Public

Prototype

```
static LStdControl* CreateFromCNTL(
     ResIDT inCNTLid,
     MessageT inValueMessage,
     ResIDT inTextTraitsID,
     LView *inSuperView);
```

Parameters This method has the following parameters:

`ResIDT`	`inCNTLid`	The resource ID of the CTNL resource.
`MessageT`	`inValueMessage`	The message that the control should send when it is changed.
`ResIDT`	`inTextTraitsID`	The resource ID of the text traits to use for drawing text.
`LView*`	`inSuperView`	The pointer to the SuperView that the control will belong to.

Return None

DisableSelf()

Purpose Disable a standard control.

Access Virtual, Protected

Prototype

```
virtual void DisableSelf();
```

Parameters None

Return None

DrawSelf()

Purpose This method is an override of the base class method DrawSelf() in LPane. It draws the standard control.

EnableSelf()

Purpose Enable a StdControl

Access Virtual, Protected

Prototype

```
virtual void EnableSelf();
```

Parameters None

Return None

FindHotSpot()

Purpose This method is an override of the base class method FindHotSpot() in LControl. It determines which hot spot, if any, contains the specified point.

FocusDraw()

Purpose This method is an override of the base class method FocusDraw() in LPane. It prepares for drawing in the Pane, and signals if the Pane has no SuperView.

GetDescriptor()

Purpose This method is an override of the base class method GetDescriptor() in LPane. It returns the Descriptor, which is the Title, of a standard control.

GetMacControl()

Purpose This method returns the value of mMacControlH.

Access Inline, Public

Prototype
```
ControlHandle GetMacControl() const;
```

Parameters None

Return The `ControlHandle` held in mMacControlH.

GetTrackingControl()

Purpose This method returns the value of sTrackingControl.

Access Static, Inline, Public

Prototype
```
static LStdControl* GetTrackingControl();
```

Parameters None

Return A pointer to an `LStdControl`, containing the value of sTrackingControl.

HideSelf()

Purpose This method is an override of the base class method GetDescriptor() in LPane. It hides a standard control.

HotSpotAction()

Purpose This method is an override of the base class method HotSpotAction() in LControl. It makes mouse tracking easier.

HotSpotResult()

Purpose This method is an override of the base class method HotSpotAction() in LControl. It performs the result of clicking and releasing mouse inside a hot spot.

InitStdControl()

Purpose This is a private initializer which creates the Mac OS Toolbox Control.

Access Private

Prototype

```
void InitStdControl( Int16 inControlKind,
     ResIDT inTextTraitsID,
     ConstStringPtr inTitle,
     Int32 inMacRefCon);
```

Parameters This method has the following parameters:

`Int16`	`inControlKind`	The value to set for mControlKind.
`ResIDT`	`inText-TraitsID`	The resource ID of the resource containing the text traits to use for drawing text.
`Const-StringPtr`	`inTitle`	The title to use for the control.
`Int32`	`inMacRefCon`	The value to put in the Mac OS refcon storage space for the control.

Return None

MoveBy()

Purpose This method is an override of the base class method FocusDraw() in LPane. This method is an override to update the Mac ControlHandle. It moves the location of the Frame by the specified amounts.

PointInHotSpot()

Purpose This method is an override of the base class method PointInHotSpot() in LControl. It tells whether a particular point is inside a hot spot or not.

ResizeFrameBy()

Purpose This method is an override of the base class method FocusDraw() in LPane. This method is an override to update the Mac ControlHandle. It changes the Frame size by the specified amounts.

SetActionProc()

Purpose This method sets the control action proc (`ControlActionUPP`) for the control.

Access Public

Prototype

```
void SetActionProc( ControlActionUPP inActionProc
     );
```

Parameters This method has the following parameter:

`ControlActionUPP`	`inActionProc`	The procedure pointer to set the control action proc to.

Return None

SetDescriptor()

Purpose This method is an override of the base class method SetDescriptor() in LPane. This method sets the Descriptor, which is the title, of a standard control.

SetMaxValue()

Purpose This method is an override of the base class method SetMaxValue() in LControl. It sets the maximum value that a control can have.

SetMinValue()

Purpose This method is an override of the base class method SetMinValue() in LControl. It sets the minimum value that a control can have.

SetStdMinAndMax()

Purpose Sets the minimum and maximum values for the standard Mac OS control associated with this object.

Access Public

Prototype

```
void SetStdMinAndMax();
```

Parameters None

Return None

Remarks

- The Control class uses Big (32 bit) values, but the Mac Control Manager only supports Small (16 bit) values. So this object class has to map between Big and Small values.
- When either the minimum or maximum value is outside 16 bit range (below -32,768 or above 32,767), we set mUsingBigValues to `true`. In that case, you must call CalcSmallValue() and CalcBigValue() to map between the Small values used by the Mac Control Manager and the Big values used by this class.

- The mapping strategy is as follows. When using Big values, we always set the minimum value for the Mac Control to zero. There are two cases for the maximum value, depending on whether the difference between the Big Max and Min is greater than or less than 32,767. If greater, we set the Mac Control maximum to 32,767. If less, we set the Mac Control maximum to that difference. These choices simplify the math for converting between Big and Small values.
- The functions CalcSmallValue() and CalcBigValue() use the following equations to convert between Big and Small values.

```
(1) BigMax - BigMin > 32,767
SmallValue = BigValue - BigMin
---------- = -----------------
32,767  (BigMax - BigMin)
(2) BigMax - BigMin <= 32,767
SmallValue = BigValue - BigMin
```

SetThumbFunc()

Purpose This method sets the value of the mThumbFunc data member.

Access Public

Prototype

```
void SetThumbFunc( ThumbActionUPP inThumbFunc );
```

Parameters This method has the following parameter:

`ThumbAction-UPP`	`inThumbFunc`	The procedure pointer to set for the control's thumb action proc.

Return None

SetValue()

Purpose This method is an override of the base class method SetValue() in LPane. This method sets the value of a standard control. It overrides

the inherited function to map from 32 bit numbers to the 16 bit numbers supported by the Mac OS Control Manager.

ShowSelf()

Purpose This method is an override of the base class method ShowSelf() in LPane. This method shows the control.

TrackHotSpot()

Purpose This method is an override of the base class method TrackHotSpot() in LControl. It tracks the mouse while it is down after clicking in a Control hot spot, and tells whether the mouse is released within the hot spot.

ValueIsInStdRange()

Purpose Indicates whether a value for a control is within the range supported by the Mac OS Control Manager.

Access Static, Protected

Prototype

```
static Boolean ValueIsInStdRange( Int32 inValue );
```

Parameters This method has the following parameter:

`Int32`	`inValue`	The value to check against the control's range.

Return Returns `true` if the value is in the range, else returns `false`.

mMacControlH

Purpose This data member stores the handle to the Mac OS control.

Access Protected

Prototype `ControlHandle mMacControlH;`

mThumbFunc

Purpose This data member stores a pointer to the thumb action procedure for the control.

Access Protected

Prototype `ThumbActionUPP mThumbFunc;`

mControlKind

Purpose This data member stores the value for the control kind. Refer to the Mac OS system header named Controls.h for more information on these enumerations.

Access Protected

Prototype `Int16 mControlKind;`

mTextTraitsID

Purpose This data member stores the value of the resource ID to use for text traits.

Access Protected

Prototype `ResIDT mTextTraitsID;`

mUsingBigValues

Purpose This data member remembers whether we are using Big values for the control or not.

Access Protected

Prototype `Boolean mUsingBigValues;`

sTrackingControl

Purpose This data member stores the pointer to the current standard control that is being tracked.

Access Static, Protected

Prototype

```
static LStdControl* sTrackingControl;
```

LStdPopupMenu

Overview LStdPopupMenu is a PowerPlant class that is used for managing Mac OS pop-up menus.

Methods The methods in this class are:

LStdPopupMenu()	~LStdPopupMenu()
DrawSelf()	GetMacMenuH()
InitStdPopupMenu()	

Data Members There are no data members in this class.

Operation LStdPopupMenu encapsulates the details of dealing with Mac OS pop-up menus. To create a new pop-up menu, create the required resources with the Constructor resource editor as described in *The PowerPlant Book*, then use a constructor to create your pop-up menu.

Source files (Pane Classes)

`LStdControl.h`

`LStdControl.cp`

Ancestors The ancestors this class derives from are:

LAttachable	LControl	LStdControl
LBroadcaster	LPane	

LStdPopupMenu()

Purpose The constructor creates objects from the passed-in parameters.

Access Public

Prototype

```
LStdPopupMenu( const SPaneInfo &inPaneInfo,
     MessageT inValueMessage,
     Int16 inTitleOptions,
```

```
        ResIDT inMENUid,
        Int16 inTitleWidth,
        Int16 inPopupVariation,
        ResIDT inTextTraitsID,
        Str255 inTitle,
        OSType inResTypeMENU,
        Int16 inInitialMenuItem );
LStdPopupMenu( const SPaneInfo &inPaneInfo,
        MessageT inValueMessage,
        Int32 inMaxValue,
        ResIDT inTextTraitsID,
        ControlHandle inMacControlH );
LStdPopupMenu( LStream *inStream );
```

Parameters These constructors have the following parameters:

`const SPaneInfo&`	`inPaneInfo`	A reference to the SPaneInfo object that is the super view.
`MessageT`	`inValueMessage`	The message sent when the control is interacted with.
`Int32`	`inTitleOptions`	This is passed to LStdControl().
`ResIDT`	`inMENUid`	The resource ID of the resource containing the menu items.
`Int16`	`inTitleWidth`	This is passed to LStdControl().
`Int16`	`inPopupVariation`	This is passed to LStdControl().
`ResIDT`	`inTextTraitsID`	The resource ID for the resource containing the text traits to apply to the pop-up menu text strings.
`Str255`	`inTitle`	The text string indicating the title of the pop-up menu.
`OSType`	`inResTypeMENU`	This is passed to LStdControl().
`Int16`	`inInitialMenu-Item`	The number indicating which menu item should appear in the pop-up menu the first time it is drawn.

`ControlHandle`	`inMacControlH`	The handle to the Mac OS control.
`LStream*`	`inStream`	Pointer to a stream object that contains the information to create the LControl object.

~LStdPopupMenu()

Purpose The destructor destroys the object.

Access Virtual, Public

Prototype

```
virtual ~LStdPopupMenu();
```

DrawSelf()

Purpose This method draws the pop-up menu, and is an override of DrawSelf() in LStdControl.

GetMacMenuH()

Purpose Return the Mac OS MenuHandle associated with a the pop-up menu.

Access Virtual, Public

Prototype

```
virtual MenuHandle GetMacMenuH();
```

Parameters None

Return MenuHandle containing the Mac OS handle of the menu.

InitStdPopupMenu()

Purpose This is a private initializer method used by the internals of the class to set up the pop-up menu.

Access Private

Prototype `void InitStdPopupMenu( Int16 inInitialMenuItem );`

Parameters This method has the following parameter:

`Int16`	`inInitialMenuItem`	The number of the initial menu item to show in the popup menu.

Return None

LStdRadioButton

Overview LStdRadioButton is a PowerPlant class that is used for managing Mac OS radio buttons.

Methods The methods in this class are:

LStdRadioButton() HotSpotResult()

SetValue()

Data Members There are no data members in this class.

Operation This class is a very simple extension of LStdControl. A click on a radio button sets the value to one. The button then broadcasts a message that it has been clicked. The broadcast message includes a pointer to the button object that was clicked.

Source files (Pane Classes)

`LStdControl.h`

`LStdControl.cp`

Ancestors The ancestors this class derives from are:

LAttachable LControl LStdControl

LBroadcaster LPane

See Also LRadioGroup

LStdRadioButton()

Purpose The constructor creates objects from the passed-in parameters.

Access Public

Prototype

```
LStdRadioButton();
```

```
LStdRadioButton( const SPaneInfo &inPaneInfo,
     MessageT inValueMessage,
     Int32 inValue,
     ResIDT inTextTraitsID,
     Str255 inTitle );
LStdRadioButton( const SPaneInfo &inPaneInfo,
     MessageT inValueMessage,
     Int32 inValue,
     ResIDT inTextTraitsID,
     ControlHandle inMacControlH );
LStdRadioButton(const LStdRadioButton &inOriginal);
LStdRadioButton( LStream *inStream );
```

Parameters These constructors have the following parameters:

`const SPaneInfo&`	`inPaneInfo`	A reference to the SPaneInfo object that is the super view.
`MessageT`	`inValueMessage`	The message sent when the control is interacted with.
`ResIDT`	`inMENUid`	The resource ID of the menu items.
`Int16`	`inValue`	The value to set for the control, must be 1 or 0. One means the control is on, zero means off.
`ResIDT`	`inTextTraitsID`	The resource ID of the resource containing the traits to apply to text strings.
`Str255`	`inTitle`	The text string to display next to the radio button.
`ControlHandle`	`inMacControlH`	This is the Mac OS control handle for the radio button.
`LStream*`	`inStream`	A pointer to a stream object that contains the information to create the LControl object.

HotSpotResult()

Purpose Respond to a click in a the radio button. This method is an override of HotSpotResult() in LStdControl. It calls SetValue() with the `Button_On` value.

In the Mac interface, clicking on a radio button always turns it on (or leaves it on). The standard way to turn off a radio button is to turn on another one in the same radio group. A radio group will normally be a Listener of a radio button.

SetValue()

Purpose Set the value of the radio button. This method is an override of SetValue() in LStdControl. It first calls the base class method, then broadcasts a message so that other buttons in the radio group (if present) can be turned off.

LStr255

Overview LStr255 is a PowerPlant class that is used for Pascal-style strings, having a maximum of 255 characters.

Methods The methods in this class are:

LStr255()	LStr255(const LStr255&)
LStr255(const LString&)	LStr255(const unsigned char*)
LStr255(unsigned char)	LStr255(char)
LStr255(const char*)	LStr255(const void*,unsigned char)
LStr255(char**)	LStr255(short,short)
LStr255(long)	LStr255(long double,signed char,short)
LStr255(unsigned long)	operator=()

Data Members The data members in this class are:

mString

Operation Use these constructors to create and convert strings into 255-character Pascal strings.

Source files (Support Classes)

`LString.h`

`LString.cp`

Ancestors LString

LStr255()

Purpose This is the default constructor.

Access Public

Prototype `LStr255();`

Parameters None

LStr255(const LStr255&)

Purpose .Copy constructor.

Access Public

Prototype `LStr255( const LStr255& inOriginal );`

Parameters This constructor takes the following parameter:

`const LStr255&`	`inOriginal`	The string to copy.

LStr255(const LString&)

Purpose Copy constructor

Access Public

Prototype `LStr255( const LString& inOriginal );`

Parameters This constructor takes the following parameter:

`const LString&`	`inOriginal`	The string to copy.

LStr255(const unsigned char*)

Purpose Constructor to convert from a pointer to a string.

Access Public

Prototype `LStr255( ConstStringPtr inStringPtr );`

Parameters This constructor takes the following parameter:

`ConstStringPtr`	`inStringPtr`	The pointer to convert.

LStr255(unsigned char)

Purpose Constructor to make a string from a single character.

Access Public

Prototype
```
LStr255( Uchar inChar );
```

Parameters This constructor takes the following parameter:

`Uchar`	`inChar`	The character to convert.

LStr255(char)

Purpose Constructor to make a string from a single character.

Access Public

Prototype
```
LStr255( char inChar );
```

Parameters This constructor takes the following parameter:

`char`	`inChar`	The character to convert.

LStr255(const char*)

Purpose Constructor to make a string from a C string (null terminated).

Access Public

Prototype
```
LStr255( const char* inCString );
```

Parameters This constructor takes the following parameter:

`const char*`	`inCString`	The string to convert.

LStr255(const void*,unsigned char)

Purpose Constructor to make a string from a pointer and length.

Access Public

Prototype
```
LStr255( const void* inPtr, Uint8 inLength );
```

Parameters This constructor takes the following parameters:

`const void*`	`inPtr`	The pointer to convert.
`Uint8`	`inLength`	The length of the string data in the pointer.

LStr255(char**)

Purpose Constructor to make a string from the data in a handle.

Access Public

Prototype
```
LStr255( Handle inHandle );
```

Parameters This constructor takes the following parameter:

`Handle`	`inHandle`	The handle to convert.

LStr255(short,short)

Purpose Constructor to make a string from a resource.

Access Public

Prototype
```
LStr255( ResIDT inResID, Int16 inIndex );
```

Parameters This constructor takes the following parameters:

`ResIDT`	`inResID`	The resource ID containing the string.
`Int16`	`inIndex`	The index into the string resource ID.

LStr255(long)

Purpose Constructor to make a string from a long integer.

Access Public

Prototype
```
LStr255( Int32 inNumber );
```

Parameters This constructor takes the following parameter:

`Int32`	`inNumber`	The long integer to create a string from.

LStr255(long double,signed char,short)

Purpose Assignment to create a string from floating point number.

Access Public

Prototype
```
LStr255( double_t inNumber, Int8 inStyle,
     Int16 inDigits );
```

Parameters This constructor takes the following parameters:

`double_t`	`inNumber`	The double to create a string from.
`Int8`	`inStyle`	The style to use, must be FLOATDECIMAL or FIXEDDECIMAL.
`Int16`	`inDigits`	• For FLOATDECIMAL, inDigits is the number of significant digits (should be > 0). • For FIXEDDECIMAL, inDigits is the number of digits to the right of the decimal point.

LStr255(unsigned long)

Purpose Constructor to create a string from a four character code, which is an unsigned long, the same as `ResType` and `OSType`.

Access Public

Prototype

```
LStr255( FourCharCode inCode );
```

Parameters This constructor takes the following parameter:

`FourCharCode`	`inCode`	The 4-character code to put in the string.

operator=()

Purpose The assignment operator performs typecasting on strings.

Access Public

Prototype

```
LStr255& operator=( FourCharCode inCode );
LStr255& operator=( Int32 inNumber );
LStr255& operator=( const char* inCString );
LStr255& operator=( char inChar );
LStr255& operator=( Uchar inChar );
LStr255& operator=( ConstStringPtr inStringPtr );
LStr255& operator=( const LString& inString );
```

Parameters This operator takes the following parameters:

`FourCharCode`	`inCode`	The 4-character code to put in the string.
`Int32`	`inNumber`	The number to put in the string.
`const char*`	`inCString`	The “C” string to convert.
`char`	`inChar`	The character to make a string from.
`Uchar`	`inChar`	The character to make a string from.
`Const-StringPtr`	`in-StringPtr`	The string pointer to make a string from.
`const LString&`	`inString`	The LString to make a string from.

Return A reference to an `LStr255`.

mString

Purpose String storage.

Access Private

Prototype

```
Str255 mString;
```

LString

Overview

LString is a PowerPlant class that is used for manipulating text strings.

Methods

The methods in this class are:

LString()
Append (TypeParameters)
AppendPStr()
Assign(TypeParameters)
BeginsWith(TypeParameters)
CStringLength()
CompareBytes()
CompareIgnoringCase()
CompareTo()
CopyPStr()
EndsWith()
Find()
FindWithin()
FourCharCodeToPStr()
Insert()
Length()
LoadFromPtr()
LoadFromSTRListResource()
LoadFromStringPtr()
PStrToFourCharCode()
Remove()
Replace()
ReverseFind()
SetCompareFunc()
ToolboxCompareText()
operator += (TypeParameter)
operator = (TypeParameter)
operator ()(unsigned char,unsigned char) const
operator [ ](unsigned char) const
operator [ ](unsigned char)
operator const unsigned char*() const
operator long double() const
operator long() const
operator unsigned char*()
operator unsigned long() const

Data Members The data members in this class are:

mStringPtr mCompareFunc

mMaxBytes

Operation This class implements string functionality for Pascal-style strings. It serves as a base class for LStr255 and TString.

Source files (Support Classes)

`LString.h`

`LString.cp`

LString()

Purpose This is a constructor to create a string from a max length and a string pointer. Subclasses must call this "protected" constructor to specify the maximum string size (including the length byte) and a pointer to the storage for the string

Access Public

Prototype

```
LString();
```

Parameters None

Append (TypeParameters)

Purpose These overloaded methods append characters to the end of this String.

Access Public

Prototype

```
LString& Append( const void* inPtr, Uint8 inLength
     );
LString& Append( Int32 inNumber );
LString& Append( char inChar );
```

```
LString& Append( Uchar inChar );
LString& Append( ConstStringPtr inStringPtr );
LString& Append( const LString& inString );
```

Parameters The parameters for these methods are:

`const void*`	`inPtr`	A pointer to the string to append.
`Int32`	`inNumber`	The number to append to the string.
`char`	`inChar`	The character to append to the string.
`Uchar`	`inChar`	The character to append to the string.
`ConstStringPtr`	`inStringPtr`	The string to append to the string.
`const LString&`	`inString`	The LString to append to the string.
`Uint8`	`inLength`	The length of the string to append.

Return A reference to an LString that has the appended string.

AppendPStr()

Purpose Append a Pascal string to this String.

Access Static, Public

Prototype

```
static StringPtr AppendPStr( Str255 ioBaseString,
    ConstStringPtr inAppendString,
    Int16 inDestSize );
```

Parameters The parameters for these methods are:

`Str255`	`ioBaseString`	A pointer to the string to append to.
`Const-StringPtr`	`inAppendString`	The string to append.
`Int16`	`inDestSize`	The size of the buffer.

Return A pointer to the finished String.

Assign(TypeParameters)

Purpose Assign a value to the string.

Access Public

Prototype

```
LString& Assign(FourCharCode inCode);
LString& Assign(double_t inNumber,
     Int8 inStyle,Int16 inDigits);
LString& Assign( Int32 inNumber);
LString& Assign(ResIDT inResID,Int16 inIndex);
LString& Assign( Handle inHandle);
LString& Assign(const void* inPtr,Uint8 inLength);
LString& Assign( const char* inCString);
LString& Assign( char inChar);
LString& Assign( Uchar inChar);
LString& Assign( ConstStringPtr inStringPtr);
LString& Assign(const LString& inString,
     Uint8 inStartPos,Uint8 inCount);
```

Parameters The parameters for this method are:

`FourCharCode`	`inCode`	A four characters in a `FourCharCode` to assign.
`double_t`	`inNumber`	The number to assign to the string.
`Int8`	`inStyle`	The style to use, must be FLOATDECIMAL or FIXEDDECIMAL.

`Int16`	`inDigits`	• For FLOATDECIMAL, inDigits is the number of significant digits (should be > 0). • For FIXEDDECIMAL, inDigits is the number of digits to the right of the decimal point.
`Int32`	`inNumber`	The number to assign to the string.
`ResIDT`	`inResID`	The LString to append to the string.
`Int16`	`inIndex`	The length of the string to append.
`Handle`	`inHandle`	The handle to the string to assign.
`const void*`	`inPtr`	The pointer to the string to assign.
`Uint8`	`inLength`	The length of the string.
`const char*`	`inCString`	The pointer to the "C" string to assign.
`char`	`inChar`	The character to assign to the string.
`Uchar`	`inChar`	The character to assign to the string.
`Const-StringPtr`	`in-StringPtr`	The pointer to the string to assign.
`const LString&`	`inString`	The pointer to the string to assign.
`Uint8`	`inStart-Pos`	The starting index in the string to assign from.
`Uint8`	`inCount`	The character count to put in the string.

Return A reference to the finished LString.

BeginsWith(TypeParameters)

Purpose Indicate whether this String begins with a specified String.

Access Public

Prototype

```
Boolean BeginsWith( const void* inPtr,
     Uint8 inLength ) const;
```

```
Boolean BeginsWith( Uchar inChar) const;
Boolean BeginsWith(
    ConstStringPtr inStringPtr) const;
Boolean BeginsWith( const LString& inString) const;
```

Parameters The parameters for this method are:

`Uint8`	`inLength`	The length of the string.
`const void*`	`inPtr`	The pointer to the string to check for.
`Uchar`	`inChar`	The character to check for at the beginning of the string.
`ConstStringPtr`	`inStringPtr`	The pointer to the string to check for.
`const LString&`	`inString`	The pointer to the string to check for.

Return A `Boolean` value of `true` if the String starts with the passed String, else `false`.

CStringLength()

Purpose Return the length of a C string, but with an upper limit of 255.

Access Static, Public

Prototype

```
static Uint8 CStringLength(const char* inCString);
```

Parameters The parameters for this method are:

`const char*`	`inCString`	The pointer to the "C" string to assign.

Return The length of the string (number of characters in it).

CompareBytes()

Purpose Simple byte-by-byte value comparison of two strings.

Access Static, Public

Prototype

```
static Int16 CompareBytes(
     const void* inLeft,
     const void* inRight,
     Uint8 inLeftLength,
     Uint8 inRightLength );
```

Parameters The parameters for this method are:

`const void*`	`inLeft`	The first string.
`const void*`	`inRight`	The second string.
`Uint8`	`inLeftLength`	The length of the first string.
`Uint8`	`inRightLength`	The length of the second string.

Return Return 1 of the Left String is greater, else return -1 if the Right String is greater. Equality will cause a return value of 0.

CompareIgnoringCase()

Purpose Case-insensitve string comparison.

Access Static, Public

Prototype

```
static Int16 CompareIgnoringCase(
     const void* inLeft,
     const void* inRight,
     Uint8 inLeftLength,
     Uint8 inRightLength );
```

Parameters The parameters for this method are:

`const void*`	`inLeft`	The first string.
`const void*`	`inRight`	The second string.
`Uint8`	`inLeftLength`	The length of the first string.
`Uint8`	`inRightLength`	The length of the second string.

Return Return 1 of the Left String is greater, else return -1 if the Right String is greater. Equality will cause a return value of 0.

CompareTo()

Purpose Compare two strings using a compare function set in mCompareFunc.

Access Protected

Prototype

```
Int16 CompareTo (const void* inPtr, Uint8 inLength)
     const;
Int16 CompareTo( Uchar inChar) const;
Int16 CompareTo(ConstStringPtr inStringPtr) const;
Int16 CompareTo( const LString& inString) const;
```

Parameters The parameters for this method are:

`const void*`	`inPtr`	The string to compare with.
`Uint8`	`inLength`	The length of the string.
`Uchar`	`inChar`	The character to compare with.
`ConstStringPtr`	`inStringPtr`	The pointer to a string to compare with.
`const LString&`	`inString`	A reference to a string to compare with.

Return The return values are specified by the compare function you supply.

CopyPStr()

Purpose Copy a Pascal String into the String.

Access Public

Prototype

```
static StringPtr CopyPStr(
     ConstStringPtr inSourceString,
     StringPtr outDestString,
     Int16 inDestSize );
```

Parameters The parameters for this method are:

`Int16`	`inDestSize`	The length of the source buffer.
`StringPtr`	`outDestString`	The destination string.
`ConstStringPtr`	`inStringPtr`	The pointer to a string to copy.

Return A pointer to the String is returned.

EndsWith()

Purpose Return whether this String ends with a specified String.

Access Public

Prototype

```
Boolean EndsWith( const void* inPtr,
     Uint8 inLength) const;
Boolean EndsWith( Uchar inChar) const;
Boolean EndsWith( ConstStringPtr inStringPtr)
     const;
Boolean EndsWith( const LString& inString) const;
```

Parameters The parameters for this method are:

`Uint8`	`inLength`	The length of the string.
`const void*`	`inPtr`	The pointer to the string to check for.
`Uchar`	`inChar`	The character to check for at the beginning of the string.
`ConstStringPtr`	`inStringPtr`	The pointer to the string to check for.
`const LString&`	`inString`	The pointer to the string to check for.

Return A `Boolean` value of `true` if the String starts with the passed String, else `false`.

Find()

Purpose Find a substring within a String.

Access Public

Prototype

```
Uint8 Find( const void* inPtr,
     Uint8 inLength, Uint8 inStartPos ) const;
Uint8 Find( Uchar inChar,
     Uint8 inStartPos = 1 ) const;
Uint8 Find( ConstStringPtr inStringPtr,
     Uint8 inStartPos = 1 ) const;
Uint8 Find( const LString& inString, Uint8
     inStartPos = 1 ) const;
```

Parameters The parameters for this method are:

`Uint8`	`inLength`	The length of the string.
`const void*`	`inPtr`	The pointer to the string to check for.

`Uint8`	`inStartPos`	The starting position in the string to search from.
`ConstStringPtr`	`inStringPtr`	The pointer to the string to check for.

Return Return the index of where the substring starts within this String.

FindWithin()

Purpose Return the index of where this String starts within another String.

Access Public

Prototype

```
Uint32 FindWithin( const void* inPtr,
     Uint32 inLength, Uint32 inStartPos ) const;
Uint8 FindWithin(Uchar inChar,
     Uint8 inStartPos = 1 ) const;
Uint8 FindWithin( ConstStringPtr inStringPtr, Uint8
     inStartPos = 1 ) const;
Uint8 FindWithin( const LString& inString, Uint8
     inStartPos = 1 ) const;
```

Parameters The parameters for this method are:

`Uint8`	`inLength`	The length of the string.
`const void*`	`inPtr`	The pointer to the string to check for.
`Uint8`	`inStartPos`	The starting position in the string to search from.
`ConstStringPtr`	`inStringPtr`	The pointer to the string to check for.

Return Returns 0 if the substring is not in the String. `inStartPos` is the index within inPtr at which to begin searching. The default value is 1. Value 0 is not valid.

Remarks To find the next occurrence, set `inStartPos` to one plus the index returned by the last `FindWithin` call.

FourCharCodeToPStr()

Purpose Convert a four character code to a Pascal string and return a pointer to the string.

Access Static, Public

Prototype

```
static StringPtr FourCharCodeToPStr(
     FourCharCode inCode,
     StringPtr outString );
```

Parameters The parameters for this method are:

`FourCharCode`	`inCode`	The four-character value to put in the string.
`StringPtr`	`outString`	The pointer to the string.

Return Returns a pointer to the string.

Remarks `outString` must point to a buffer (usually a string) that can hold the resulting string, including the length byte. Therefore, the buffer must be at least 5-bytes large. Usually, you will allocate an array of 5 `unsigned char`.

Insert()

Purpose

Access Public

Prototype

```
LString& Insert( const void* inPtr,
     Uint8 inLength, Uint8 inAtIndex);
LString& Insert( Uchar inChar, Uint8 inAtIndex);
LString& Insert( ConstStringPtr inStringPtr, Uint8
     inAtIndex);
```

```
LString& Insert( const LString& inString, Uint8
     inAtIndex);
```

Parameters The parameters for this method are:

`Uint8`	`inLength`	The length of the string.
`const void*`	`inPtr`	The pointer to the string to insert.
`Uint8`	`inAtIndex`	The starting position in the string to insert from.
`Uint8`	`inChar`	The character to insert in the string.
`ConstStringPtr`	`inStringPtr`	The pointer to the string to insert.
`const LString&`	`inString`	A reference to the string to insert.

Return A reference to the finished string.

Length()

Purpose Indicate the String length.

Access Public

Prototype

```
Uint8 Length() const;
```

Parameters None

Return Returns the number of characters in the String.

LoadFromPtr()

Purpose Fill in the String given a pointer and a byte count.

Access Protected

Prototype

```
void LoadFromPtr( const void* inPtr,
     Uint8 inLength );
```

Parameters The parameters for this method are:

`Uint8`	`inLength`	The length of the string.
`const void*`	`inPtr`	The pointer to the string to insert.

Return None

LoadFromSTRListResource()

Purpose Fill in the String from a STR resource.

Access Protected

Prototype

```
void LoadFromSTRListResource( ResIDT inResID,
     Int16 inIndex);
void LoadFromSTRResource( ResIDT inResID );
```

Parameters The parameters for this method are:

`Uint8`	`inLength`	The index of the string within the resource.
`ResIDT`	`inResID`	The resource ID containing the string.

Return None

LoadFromStringPtr()

Purpose This method calls LoadFromPtr().

Access Public

Prototype

```
void LoadFromStringPtr( ConstStringPtr
     inStringPtr);
```

Parameters The parameter for this method is:

`ConstStringPtr`	`inStringPtr`	The pointer to the string to use.

Return None

PStrToFourCharCode()

Purpose Convert a Pascal string to a four character code.

Access Static, Public

Prototype

```
static void PStrToFourCharCode(
     ConstStringPtr inString,
     FourCharCode &outCode );
```

Parameters The parameters for this method are:

`ConstStringPtr`	`inString`	The pointer to the string to use.
`FourCharCode&`	`outCode`	The four-character code extracted from the string.

Return None

Remove()

Purpose Remove a given number of bytes from this String starting at the specified index.

Access Public

Prototype

```
LString& Remove(
     Uint8 inStartPos,
     Uint8 inCount );
```

Parameters The parameters for this method are:

`Uint8`	`inStartPos`	The starting index in the string.
`Uint8`	`inCount`	The character count to remove.

Return A reference to the String.

Replace()

Purpose Replace a portion of this String with a new value.

Access Public

Prototype

```
LString& Replace( Uint8 inStartPos, Uint8 inCount,
     const void* inPtr, Uint8 inLength );
LString& Replace( Uint8 inStartPos, Uint8 inCount,
     Uchar inChar );
LString& Replace( Uint8 inStartPos, Uint8 inCount,
     ConstStringPtr inStringPtr );
LString& Replace( Uint8 inStartPos, Uint8 inCount,
     const LString& inString );
```

Parameters The parameters for this method are:

`Uint8`	`inStartPos`	The starting index in the string.
`Uint8`	`inCount`	The character count to remove.
`const void*`	`inPtr`	The pointer to the string.
`Uint8`	`inLength`	The length of the string.
`const LString&`	`inString`	The reference to the string.
ConstStringPtr	inStringPtr	The pointer to the string.
Uchar	inChar	The character to replace.

Return A reference to the String.

ReverseFind()

Purpose Find where a substring starts within this String, searching from the end of the String towards the beginning of the String.

Access Public

Prototype

```
Uint8 ReverseFind( const void* inPtr, Uint8
     inLength, Uint8 inStartPos ) const;
Uint8 ReverseFind( Uchar inChar, Uint8 inStartPos
     = 255 ) const;
Uint8 ReverseFind( ConstStringPtr inStringPtr,
     Uint8 inStartPos = 255 ) const;
Uint8 ReverseFind( const LString& inString, Uint8
     inStartPos = 255 ) const;
Uint32 ReverseFindWithin( const void* inPtr,
     Uint32 inLength, Uint32 inStartPos ) const;
Uint8 ReverseFindWithin( Uchar inChar, Uint8
     inStartPos = 1 ) const;
Uint8 ReverseFindWithin( ConstStringPtr
     inStringPtr, Uint8 inStartPos = 255 ) const;
Uint8 ReverseFindWithin( const LString& inString,
     Uint8 inStartPos = 255 ) const;
```

Parameters The parameters for this method are:

`Uint8`	`inStartPos`	The starting index in the string.
`const void*`	`inPtr`	The pointer to the string.
`Uint8`	`inLength`	The length of the string.
`const LString&`	`inString`	The reference to the string.
`ConstStringPtr`	`inStringPtr`	The pointer to the string.
`Uchar`	`inChar`	The character to find.

Return The index where the item was found.

SetCompareFunc()

Purpose Sets the value of mCompareFunc to point to a function used for comparing strings.

Access Public

Prototype

```
void SetCompareFunc( CompareFunc inCompareFunc );
```

Parameters The parameters for this method are:

`CompareFunc`	`inCompareFunc`	The pointer to the compare function.

Return None

ToolboxCompareText()

Purpose This method is a String comparison function that uses the Toolbox `CompareText()` routine.

Access Static, Public

Prototype

```
static Int16 ToolboxCompareText(
    const void* inLeft,
    const void* inRight,
    Uint8 inLeftLength,
    Uint8 inRightLength );
```

Parameters The parameters for this method are:

`const void*`	`inLeft`	The first string.
`const void*`	`inRight`	The second string.
`Uint8`	`inLeftLength`	The length of the first string.
`Uint8`	`inRightLength`	The length of the second string.

Return This is the same as the return value for `CompareText()` in the Mac OS Toolbox.

operator += (TypeParameter)

Purpose This is an operator overload for the += operator. It calls the Append (TypeParameters) method to append a String to another String.

TypeParameter long, char, unsigned char, const unsigned char*, const LString&

Access Public

Prototype `LString& operator+=(`*TypeParameter* `);`

Parameters The specified TypeParameters may be input to this operator.

Return A reference to the String.

operator = (TypeParameter)

Type Parameter unsigned long, long, const char*, char, unsigned char, const unsigned char*, const LString&

Purpose This is an operator overload for the = (assignment) operator. It calls the Assign(TypeParameters) method to assign a String to another String.

TypeParameter unsigned long, long, const char*, char, unsigned char, const unsigned char*, const LString&

Access Public

Prototype `LString& operator=(`*TypeParameter* `);`

Parameters The specified TypeParameters may be input to this operator.

Return A reference to the String.

operator ()(unsigned char,unsigned char) const

Purpose Return a `LStr255` String object containing the specified substring of this String.

Access Public

Prototype

```
LStr255 operator()(
     Uint8 inStartPos,
     Uint8 inCount ) const;
```

Parameters The parameters for this operator are:

`Uint8`	`inStartPos`	The start position within the string.
`Uint8`	`inCount`	The length of the string to return.

Return An `LStr255`

operator [ ](unsigned char) const

Purpose Return a character at a specified position in the String.

Access Public

Prototype

```
const Uchar& operator[]( Uint8 inPosition ) const;
```

Parameters The parameter for this operator is:

`Uint8`	`inPosition`	The start position within the string.

Return A reference to the unsigned character.

operator [ ](unsigned char)

Purpose Access individual characters in a String, in an array-like fashion.

Access Inline, Public

Prototype `Uchar& operator[]( Uint8 inPosition );`

Parameters The parameter for this operator is:

Uint8	inPosition	The start position within the string.

Return A reference to the unsigned character.

operator const unsigned char*() const

Purpose Retrieve the value of mStringPtr.

Access Inline, Public

Prototype `operator ConstStringPtr() const;`

Parameters None

Return A pointer to the String held in mStringPtr.

operator long double() const

Purpose This operator provides a conversion to a floating point number.

Access Public

Prototype `operator double_t() const;`

Parameters None

Return A `double_t` that represents the String.

operator long() const

Purpose This operator provides a conversion to a long integer.

Access Public

Prototype `operator Int32() const;`

Parameters None

Return A `Int32` that represents the String.

operator unsigned char*()

Purpose This operator provides a conversion to a `char*` pointer.

Access Public

Prototype `operator StringPtr();`

Parameters None

Return A `StringPtr` that represents the String.

operator unsigned long() const

Purpose This operator provides a conversion to a four character code.

Access Public

Prototype `operator FourCharCode() const;`

Parameters None

Return A `FourCharCode` that represents the String.

mStringPtr

Purpose The String pointer.

Access Protected

Prototype `StringPtr mStringPtr;`

mCompareFunc

Purpose The compare function pointer for the String.

Access Protected

Prototype

```
CompareFunc mCompareFunc;
```

mMaxBytes

Purpose The maximum number of bytes for the String.

Access Protected

Prototype

```
Uint16 mMaxBytes;
```

LSubOverlapView

Overview LSubOverlapView is a PowerPlant class that is a container view for Panes which might overlap.

Methods The methods in this class are:

LSubOverlapView() ~LSubOverlapView()

FocusDraw()

Data Members There are no data member in this class.

Operation This class inherits from the chain of classes: LPane, LView and LAttachable. This class helps manage panes that overlap.

Source files (Pane Classes)

`LSubOverlapView.h`

`LSubOverlapView.cp`

Ancestors LAttachable

LPane

LView

LSubOverlapView()

Purpose The constructors create the objects from the passed-in parameters.

Access Public

Prototype

```
LSubOverlapView();
LSubOverlapView( LStream *inStream );
```

Parameters The parameter for these constructors is:

`LStream*`	`inStream`	The stream to read from to create the object.

~LSubOverlapView()

Purpose The destructor destroys the object.

Access Virtual, Public

Prototype

```
virtual ~LSubOverlapView();
```

FocusDraw()

Purpose Set up coordinates system and clipping region. Panes rely on their superview to focus them. When a Pane requests focus for drawing, a SubOverlapView sets the clipping region to revealed portion of that Pane's Frame minus the Frames of all sibling Panes that are in front of that Pane. This is an override of FocusDraw() in LPane.

LTabGroup

Overview

LTabGroup is a PowerPlant class that is used for managing tab key navigation between two or more edit fields in a view. Mac OS Human Interface Guidelines suggest that typing the Tab key should advance the text entry cursor from one field to the next.

Methods

The methods in this class are:

LTabGroup()	~LTabGroup()
BeTarget()	GetOnDutySub()
HandleKeyPress()	RotateTarget()

Data Members

There are no data members is this class.

Operation

A Tab Group switches the Target amongst its SubCommanders in response to Tab and Shift-Tab key presses. A SubCommander is responsible for passing up Tab key presses to its SuperCommander if it wants to be usuable with a TabGroup.

By default, the first SubCommander of a TabGroup will be the Target when the Window containing the TabGroup is activated. If there is more than one TabGroup in a Window, then the last TabGroup will be the one on duty.

To make a particular Commander be the Target when its Window is activated, callSetLatentSub() for that LCommander.

Source files

(Support Classes)

`LTabGroup.h`

`LTabGroup.cp`

Ancestors

LAttachable

LCommander

LTabGroup()

Purpose The constructor creates objects from the passed-in parameters.

Access Public

Prototype

```
LTabGroup();
LTabGroup( LStream* inStream );
LTabGroup( LCommander *inStream );
```

Parameters The parameters for these constructors are:

`LStream*`	`inStream`	This is unused.
`LCommander*`	`inSuper`	Pointer to the SuperCommander.

~LTabGroup()

Purpose The destructor destroys the object.

Access Virtual, Public

Prototype

```
virtual ~LTabGroup();
```

BeTarget()

Purpose TabGroup has become the Target. This is an override of BeTarget() in LCommander.

GetOnDutySub()

Purpose Return the on duty SubCommander of a TabGroup.

Access Virtual, Protected

Prototype

```
virtual LCommander* GetOnDutySub();
```

Parameters None

Return A pointer to the LCommander on duty.

HandleKeyPress()

Purpose Tab switches the Target to the next item in the TabGroup. Shift-Tab to the previous item.

All other keystrokes (and Tabs with modifiers other than Shift) get passed up. This is an override of HandleKeyPress() in LCommander.

RotateTarget()

Purpose Switch Target to another SubCommander, either the one before or after the current one.

Access Virtual, Public

Prototype

```
virtual void RotateTarget( Boolean inBackward );
```

Parameters The parameters for these constructors are:

`Boolean`	`inBackward`	This indicates whether to rotate backward or forward in the sequence.

Return None

LTable

Overview

LTable is a PowerPlant class that is used for display and managemnet of tabular data: rows and columns of rectangular cells in a two-dimensional grid.

Methods

The methods in this class are:

LTable()	~LTable()
ActivateSelf()	ClickCell()
ClickSelf()	DeactivateSelf()
DrawCell()	DrawSelf()
EqualCell()	FetchCellDataIndex()
FetchCellHitBy()	FetchLocalCellFrame()
GetCellData()	GetSelectedCell()
GetTableSize()	HiliteCell()
InitTable()	InsertCols()
InsertRows()	IsValidCell()
RemoveCols()	RemoveRows()
SelectCell()	SetCellData()
SetCellDataSize()	SetColWidth()
SetRowHeight()	UnhiliteCell()

Data Members

The data members in this class are:

mRows	mCols
mRowHeight	mColWidth
mCellData	mSelectedCell

Operation

Refer to *The PowerPlant Book* for a detailed introduction to this class.

Source files (Pane Classes)

`LTable.h`

`LTable.cp`

See also LAttachable

LPane

LView

LTable()

Purpose The constructor creates the object.

Access Public

Prototype

```
LTable();
LTable(
     const SPaneInfo&inPaneInfo,
     const SViewInfo&inViewInfo,
     Int32 inNumberOfRows,
     Int32 inNumberOfCols,
     Int32 inRowHeight,
     Int32 inColWidth,
     Int32 inCellDataSize );
LTable( LStream*inStream );
```

~LTable()

Purpose The destructor destroys the object.

Access Virtual, Public

Prototype

```
virtual ~LTable();
```

ActivateSelf()

Purpose This is an override of ActivateSelf() in LPane().

ClickCell()

Purpose Respond to a click in the cell.

Access Virtual, Protected

Prototype

```
virtual void ClickCell(
     const TableCellT&inCell,
     const SMouseDownEvent& inMouseDown );
```

Parameters The parameters for this method are:

`const TableCellT&`	`inCell`	The cell.
`const SMouseDownEvent&`	`inMouseDown`	The mouse down event info.

Return None

ClickSelf()

Purpose This is an override of ClickSelf() in LPane().

DeactivateSelf()

Purpose This is an override of DeactivateSelf() in LPane().

DrawCell()

Purpose Draw the contents of a cell.

Access VIrtual, Protected

Prototype `virtual void DrawCell(const TableCellT&inCell);`

Parameters The parameters for this method are:

`const TableCellT&`	`inCell`	The cell.

Return None

DrawSelf()

Purpose This is an override of DrawSelf() in LPane().

EqualCell()

Purpose This method tests 2 cells for equality.

Access Public

Prototype

```
Boolean EqualCell(
    const TableCellT&inCellA,
    const TableCellT&inCellB) const;
```

Parameters The parameters for this method are:

`const TableCellT&`	`inCellA`	The first cell.
`const TableCellT&`	`inCellB`	The other cell.

Return Returns true if equal, else false.

FetchCellDataIndex()

Purpose Retrieves the index of a cell containing the data indicated.

Access Protected

Prototype

```
Int32 FetchCellDataIndex(
    const TableCellT&inCell );
```

Parameters The parameters for this method are:

`const TableCellT&`	`inCell`	The cell.

Return Returns the cell index.

FetchCellHitBy()

Purpose Find a cell hit by a certain point.

Access Virtual, Protected

Prototype

```
virtual void FetchCellHitBy(
    const SPoint32&inImagePt,
    TableCellT&outCell );
```

Parameters The parameters for this method are:

`const SPoint32&`	`inImagePt`	The point.
`const TableCellT&`	`outCell`	The cell.

Return None

FetchLocalCellFrame()

Purpose See if a cell intersects a given frame.

Access Virtual, Protected

Prototype

```
virtual Boolean FetchLocalCellFrame(
    const TableCellT&inCell,
    Rect    &outCellFrame);
```

Parameters The parameters for this method are:

`Rect&`	`outCellFrame`	The frame rect for the cell.
`const TableCellT&`	`inCell`	The cell.

Return Return true if there is an intersection, else return false.

GetCellData()

Purpose Retrieve the data from a cell.

Access Virtual, Public

Prototype

```
virtual void GetCellData(
    const TableCellT&inCell,
    void    *outData);
```

Parameters The parameters for this method are:

`const TableCellT&`	`inCell`	The cell.
`void*`	`outData`	The cell data.

Return None

GetSelectedCell()

Purpose Retrieve the contents of a selected cell.

Access Public

Prototype

```
void GetSelectedCell(TableCellT&outCell) const;
```

Parameters The parameters for this method are:

`TableCellT&`	`outCell`	The cell.

Return None

GetTableSize()

Purpose Retrieve the size of the table.

Access Public

Prototype

```
void GetTableSize(
     TableIndexT&outRows,
     TableIndexT&outCols ) const;
```

Parameters The parameters for this method are:

`TableCellT&`	`outRows`	The rows.
`TableCellT&`	`outCols`	The columns.

Return None

HiliteCell()

Purpose Hilite a given cell.

Access Virtual, Protected

Prototype

```
virtual void HiliteCell(
     const TableCellT& inCell);
```

Parameters The parameters for this method are:

`TableCellT&`	`inCell`	The cell.

Return None

InitTable()

Purpose

Access Private

Prototype

```
void InitTable(
     Int32 inNumberOfRows,
     Int32 inNumberOfCols,
     Int32 inRowHeight,
     Int32 inColWidth,
     Int32 inCellDataSize);
```

Parameters The parameters for this method are:

`Int32`	`inNumberOfRows`	The number of rows.
`Int32`	`inNumberOfCols`	The number of columns
`Int32`	`inRowHeight`	The row height.
`Int32`	`inColWidth`	The column width.
`Int32`	`inCellDataSize`	The cell data size.

Return None

InsertCols()

Purpose Insert columns after a specified column.

Access Virtual, Public

Prototype

```
virtual void InsertCols(
     Int32inHowMany,
     TableIndexTinAfterCol,
     void*inCellData );
```

Parameters The parameters for this method are:

`Int32`	`inHowMany`	The number of columns.

`TableIndexT`	`inAfterCol`	The column to insert after.
`void*`	`inCellData`	The cell data.

Return None

InsertRows()

Purpose Insert rows after a specified row.

Access Virtual, Public

Prototype

```
virtual void InsertRows(
     Int32inHowMany,
     TableIndexTinAfterRow,
     void*inCellData);
```

Parameters The parameters for this method are:

`Int32`	`inHowMany`	The number of rows.
`TableIndexT`	`inAfterRow`	The row to insert after.
`void*`	`inCellData`	The cell data.

Return None

IsValidCell()

Purpose Indicate whether a cell is valid or not.

Access Public

Prototype

```
Boolean IsValidCell(
     const TableCellT&inCell) const;
```

Parameters The parameters for this method are:

`TableCellT&`	`inCell`	The row to insert after.

Return Returns true if valid, else returns false.

RemoveCols()

Purpose Remove columns starting after a given column.

Access Virtual, Public

Prototype

```
virtual void RemoveCols(
     Int32inHowMany,
     TableIndexTinFromCol );
```

Parameters The parameters for this method are:

`Int32`	`inHowMany`	The number of columns to remove.
`TableIndexT`	`inFromCol`	The starting column.

Return None

RemoveRows()

Purpose Remove rows starting after a given row.

Access Virtual, Public

Prototype

```
virtual void RemoveRows(
     Int32inHowMany,
     TableIndexTinFromRow);
```

Parameters The parameters for this method are:

`Int32`	`inHowMany`	The number of rows to remove.
`TableIndexT`	`inFromRow`	The starting row.

Return None

SelectCell()

Purpose Select a given cell.

Access Virtual, Public

Prototype

```
virtual void SelectCell(
     const TableCellT&inCell );
```

Parameters The parameters for this method are:

`const TableCellT&`	`inCell`	The cell.

Return None

SetCellData()

Purpose Set cell data.

Access Virtual, Public

Prototype

```
virtual void SetCellData(
     const TableCellT&inCell,
     void    *inData );
```

Parameters The parameters for this method are:

`const TableCellT&`	`inCell`	The cell.
`void*`	`inData`	The data to set.

Return None

SetCellDataSize()

Purpose Set the cell data size.

Access Public

Prototype

```
void SetCellDataSize( Int32inCellDataSize );
```

Parameters The parameters for this method are:

`Int32`	`inCellDataSize`	The data to set.

Return None

SetColWidth()

Purpose Set the column width.

Access Virtual, Public

Prototype

```
virtual void SetColWidth(
     Int16inWidth,
     TableIndexT inFrom,
     TableIndexT inTo);
```

Parameters The parameters for this method are:

`Int16`	`inWidth`	The width to set.
`TableIndexT`	`inFrom`	The starting index.
`TableIndexT`	`inTo`	The ending index.

Return None

SetRowHeight()

Purpose Set the row height.

Access Virtual, Public

Prototype

```
virtual void SetRowHeight(
     Int16inHeight,
     TableIndexT inFrom,
     TableIndexT inTo );
```

Parameters The parameters for this method are:

`Int16`	`inHeight`	The height to set.
`TableIndexT`	`inFrom`	The starting index.
`TableIndexT`	`inTo`	The ending index.

Return None

UnhiliteCell()

Purpose Unhilite a cell.

Access Virtual, Protected

Prototype

```
virtual void UnhiliteCell(
     const TableCellT& inCell);
```

Parameters The parameters for this method are:

`const TableCellT&`	`inCell`	The cell.

Return None

mRows

Purpose The number of rows.

Access Protected

Prototype
```
TableIndexTmRows;
```

mCols

Purpose The number of columns.

Access Protected

Prototype
```
TableIndexTmCols;
```

mRowHeight

Purpose The row height.

Access Protected

Prototype
```
Int32mRowHeight;
```

mColWidth

Purpose The column width.

Access Protected

Prototype
```
Int32mColWidth;
```

mCellData

Purpose The cell data array.

Access Protected

Prototype `LArray*mCellData;`

mSelectedCell

Purpose The selected cell.

Access Protected

Prototype `TableCellTmSelectedCell;`

LTableArrayStorage

Overview LTableArrayStorage is a concrete implementation of LTableStorage to support a table where data is stored in an array.

Methods The methods in this class are:

LTableArrayStorage()	~LTableArrayStorage()
FindCellData()	GetCellData()
GetStorageSize()	InsertCols()
InsertRows()	RemoveCols()
RemoveRows()	SetCellData()

Data Members The data members in this class are:

mDataArray	mOwnsArray

Operation LTableArrayStorage adds no new member functions, but implements every fucntion for LTableStorage.

Source files (Table Classes)

`LTableArrayStorage.h`

`LTableArrayStorage.cp`

See also LTableStorage

LArray

LTableArrayStorage()

Purpose The constructor creates the object.

Access Public

Prototype For cells with the same size data, this creates an LArray object with items equal to the indicated size.

```
LTableArrayStorage( LTableView*inTableView,
     Uint32inDataSize );
This is a user-specified subclass of LArray.
LTableArrayStorage(
     LTableView*inTableView,
     LArray*inDataArray );
```

Parameters None

~LTableArrayStorage()

Purpose The destructor destroys the object.

Access Virtual, Public

Prototype

```
virtual ~LTableArrayStorage();
```

FindCellData()

Purpose This is an override of FindCellData() in LTableStorage.

GetCellData()

Purpose This is an override of GetCellData() in LTableStorage.

GetStorageSize()

Purpose This is an override of GetStorageSize() in LTableStorage.

InsertCols()

Purpose This is an override of InsertCols() in LTableStorage.

InsertRows()

Purpose This is an override of InsertRows() in LTableStorage.

RemoveCols()

Purpose This is an override of RemoveCols() in LTableStorage.

RemoveRows()

Purpose This is an override of RemoveRows() in LTableStorage.

SetCellData()

Purpose This is an override of SetCellData() in LTableStorage.

mDataArray

Purpose The data array.

Access Protected

Prototype
```
LArray     *mDataArray;
```

mOwnsArray

Purpose Determines whether the array is destroyed when the LTableArrayStorage object is destroyed.

Access Protected

Prototype
```
Boolean    mOwnsArray;
```

LTableGeometry

Overview

LTableGeometry is an abstract PowerPlant class that specifies the interface for the geometry helper objects. These functions provide behaviors to maintain the location, width, and height of each cell in a table.

Methods

The methods in this class are:

LTableGeometry()
~LTableGeometry()
GetColHitBy()
GetColWidth()
GetImageCellBounds()
GetRowHeight()
GetRowHitBy()
GetTableDimensions()
InsertCols()
InsertRows()
RemoveCols()
RemoveRows()
SetColWidth()
SetRowHeight()

Data Members

The data members in this class are:

mTableView

Operation

All the methods in this class are either pure virtual or empty. These methods form the basis for interacting with a table's geometry in PowerPlant. Most of the time you will not need to concern yourself with these methods. The PowerPlant table classes call these methods to get or set required data. In general, you should call the table methods, not the related geometry methods.

Source files

(Table Classes)

`UTableHelpers.h`

LTableGeometry()

Purpose

The constructor creates an object.

Access Public

Prototype
```
LTableGeometry();
```

~LTableGeometry()

Purpose The destructor destroys the object.

Access Virtual, Public

Prototype
```
virtual  ~LTableGeometry();
```

GetColHitBy()

Purpose Returns the index of a column that contains a point.

Access Pure Virtual, Public

Prototype
```
virtual TableIndexTGetColHitBy(
     const SPoint32&inImagePt )
     const = 0;
```

Parameters This method has the following parameters:

`SPoint32&`	`inImagePt`	A reference to the point.

Return Returns the index for the table column containing the point.

GetColWidth()

Purpose Retrieve the column width for a specified column.

Access Pure Virtual, Public

Prototype
```
virtual Uint16GetColWidth(
     TableIndexTinCol ) const = 0;
```

Parameters This method has the following parameters:

`TableIndexT&`	`inCol`	A reference to the column.

Return The column.

GetImageCellBounds()

Purpose Provides bounds of cell in image coordinates.

Access Pure Virtual, Public

Prototype

```
virtual voidGetImageCellBounds(
     const STableCell&inCell,
     Int32   &outLeft,
     Int32   &outTop,
     Int32   &outRight,
     Int32   &outBottom ) const = 0;
```

Parameters This method has the following parameters:

`STableCell&`	`inCell`	A reference to the cell.
`Int32`	`outLeft`	The left coordinate.
`Int32`	`outTop`	The top coordinate.
`Int32`	`outRight`	The right coordinate.
`Int32`	`outBottom`	The bottom coordinate.

Return None

GetRowHeight()

Purpose Determine the row height.

Access Pure Virtual, Public

Prototype

```
virtual Uint16GetRowHeight(
     TableIndexTinRow ) const = 0;
```

Parameters This method has the following parameters:

`TableIndexT&`	`inRow`	A reference to the row.

Return The row height.

GetRowHitBy()

Purpose Returns the index of a row that contains a point.

Access Pure Virtual, Public

Prototype

```
virtual TableIndexTGetRowHitBy(
     const SPoint32&inImagePt) const = 0;
```

Parameters This method has the following parameters:

`SPoint32&`	`inImagePt`	A reference to the point.

Return The row index.

GetTableDimensions()

Purpose Determine the dimensions of the table.

Access Pure Virtual, Public

Prototype

```
virtual voidGetTableDimensions(
     Uint32&outWidth,
     Uint32&outHeight ) const = 0;
```

Parameters This method has the following parameters:

`Uint32&`	`outWidth`	A reference to the width.
`Uint32&`	`outHeight`	A reference to the height.

Return None

InsertCols()

Purpose Insert columns after a specified column.

Access Pure Virtual, Public

Prototype

```
virtual voidInsertCols(
     Uint32 inHowMany,
     TableIndexT inAfterCol);
```

Parameters This method has the following parameters:

`Uint32`	`inHowMany`	The number of columns.
`TableIndexT`	`inAfterCol`	The column index to insert after.

Return None

InsertRows()

Purpose Insert rows after a given row.

Access Pure Virtual, Public

Prototype

```
virtual voidInsertRows(
     Uint32 inHowMany,
     TableIndexT inAfterRow );
```

Parameters This method has the following parameters:

`Uint32`	`inHowMany`	The number of columns.
`TableIndexT`	`inAfterRow`	The row index to insert after.

Return None

RemoveCols()

Purpose Remove columns after a specified starting column.

Access Pure Virtual, Public

Prototype
```
virtual voidRemoveCols(
    Uint32inHowMany,
    TableIndexTinFromCol);
```

Parameters This method has the following parameters:

`Uint32`	`inHowMany`	The number of columns.
`TableIndexT`	`inFromCol`	The column index to start removal after.

Return None

RemoveRows()

Purpose The row to start removal at.

Access Pure Virtual, Public

Prototype
```
virtual voidRemoveRows(
    Uint32inHowMany,
    TableIndexTinFromRow);
```

Parameters This method has the following parameters:

`Uint32`	`inHowMany`	The number of columns.
`TableIndexT`	`inFromRow`	The row index to start removal after.

Return None

SetColWidth()

Purpose Set the width of a range of columns.

Access Pure Virtual, Public

Prototype
```
virtual voidSetColWidth(
    Uint16inWidth,
```

```
    TableIndexT inFromCol,
    TableIndexTinToCol) = 0;
```

Parameters This method has the following parameters:

Uint16	`inWidth`	The width of the columns.
`TableIndexT`	`inFromCol`	The starting column.
`TableIndexT`	`inToCol`	The ending column.

Return None

SetRowHeight()

Purpose Set the height on a range of rows.

Access Pure Virtual, Public

Prototype

```
virtual voidSetRowHeight(
    Uint16inHeight,
    TableIndexTinFromRow,
    TableIndexTinToRow) = 0;
```

Parameters This method has the following parameters:

Uint16	`inWidth`	The height of the rows.
`TableIndexT`	`inFromRow`	The starting row.
`TableIndexT`	`inToRow`	The ending row.

Return None

mTableView

Purpose The pointer to the table that owns this helper object.

Access Protected

Prototype

```
LTableView*mTableView;
```

LTableMonoGeometry

Overview LTableMonoGeometry is a concrete implemantation of LTableGeometry to support a table where all cells are the same size.

Methods The methods in this class are:

LTableMonoGeometry()	~LTableMonoGeometry()
GetColHitBy()	GetColWidth()
GetImageCellBounds()	GetRowHeight()
GetRowHitBy()	GetTableDimensions()
SetColWidth()	SetRowHeight()

Data Members The data members in this class are:

mColWidth	mRowHeight

Operation This class adds no new member functions, but implements every pure virtual function listed in LTableGeometry.

Source files (Table Classes)

`LTableMonoGeometry.h`

`LTableMonoGeometry.cp`

See also LTableGeometry

LTableMonoGeometry()

Purpose The constructor creates the object.

Access Public

Prototype

```
LTableMonoGeometry( LTableView*inTableView,
```

```
        Uint16 inColWidth,
        Uint16 inRowHeight );
```

Parameters This method has the following parameters:

`LTableView*`	`inTableView`	A pointer to the table.
`Uint16`	`inColWidth`	The column width.
`Uint16`	`inRowHeight`	The row height.

~LTableMonoGeometry()

Purpose The destructor destroys the object.

Access Virtual, Public

Prototype

```
virtual~LTableMonoGeometry();
```

GetColHitBy()

Purpose This is an override of GetColHitBy() in LTableGeometry.

GetColWidth()

Purpose This is an override of GetColWidth() in LTableGeometry.

GetImageCellBounds()

Purpose This is an override of GetImageCellBounds() in LTableGeometry.

GetRowHeight()

Purpose This is an override of GetRowHeight() in LTableGeometry.

GetRowHitBy()

Purpose This is an override of GetRowHitBy() in LTableGeometry.

GetTableDimensions()

Purpose This is an override of GetTableDimensions() in LTableGeometry.

SetColWidth()

Purpose This is an override of SetColWidth() in LTableGeometry.

SetRowHeight()

Purpose This is an override of SetRowHeight() in LTableGeometry.

mColWidth

Purpose The width of columns.

Access Protected

Prototype
```
Uint16      mColWidth;
```

mRowHeight

Purpose The height of rows.

Access Protected

Prototype
```
Uint16      mRowHeight;
```

LTableMultiGeometry

Overview

LTableMultiGeometry is a concrete implementation of LTableGeometry to support a table where rows and columns may vary in size.

Methods

The methods in this class are:

LTableMultiGeometry()	~LTableMultiGeometry()
GetColHitBy()	GetColWidth()
GetImageCellBounds()	GetRowHeight()
GetRowHitBy()	GetTableDimensions()
InsertCols()	InsertRows()
RemoveCols()	RemoveRows()
SetColWidth()	SetRowHeight()

Data Members

The data members in this class are:

mRowHeights	mColWidths
mDefaultRowHeight	mDefaultColWidth

Operation

LTableMultiGeometry adds no new methods, but implements every method listed in LTableGeometry. The insert and remove of rows and columns methods maintain the arrays of heights and widths for the table.

Source files

(Table Classes)

`LTableMultiGeometry.h`

`LTableMultiGeometry.cp`

See also

LTableGeometry

LTableMultiGeometry()

Purpose The constructor creates the object.

Access Public

Prototype

```
LTableMultiGeometry( LTableView*inTableView,
     Uint16 inColWidth,
     Uint16 inRowHeight );
```

Parameters This method has the following parameters:

`LTableView*`	`inTableView`	A pointer to the table.
`Uint16`	`inColWidth`	The column width.
`Uint16`	`inRowHeight`	The row height.

~LTableMultiGeometry()

Purpose The destructor destroys the object.

Access Virtual, Public

Prototype

```
virtual ~LTableMultiGeometry();
```

GetColHitBy()

Purpose This is an override of GetColHitBy() in LTableGeometry.

GetColWidth()

Purpose This is an override of GetColWidth() in LTableGeometry.

GetImageCellBounds()

Purpose This is an override of GetImageCellBounds() in LTableGeometry.

GetRowHeight()

Purpose This is an override of GetRowHeight() in LTableGeometry.

GetRowHitBy()

Purpose This is an override of GetRowHitBy() in LTableGeometry.

GetTableDimensions()

Purpose This is an override of GetTableDimensions() in LTableGeometry.

InsertCols()

Purpose This is an override of InsertCols() in LTableGeometry.

InsertRows()

Purpose This is an override of InsertRows() in LTableGeometry.

RemoveCols()

Purpose This is an override of RemoveCols() in LTableGeometry.

RemoveRows()

Purpose This is an override of RemoveRows() in LTableGeometry.

SetColWidth()

Purpose This is an override of SetColWidth() in LTableGeometry.

SetRowHeight()

Purpose This is an override of SetRowHeight() in LTableGeometry.

mRowHeights

Purpose The height of rows.

Access Protected

Prototype
```
LRunArraymRowHeights;
```

mColWidths

Purpose The width of columns.

Access Protected

Prototype
```
LRunArraymColWidths;
```

mDefaultRowHeight

Purpose The default row height.

Access Protected

Prototype
```
Uint16mDefaultRowHeight;
```

mDefaultColWidth

Purpose The default column width.

Access Protected

Prototype
```
Uint16mDefaultColWidth;
```

LTableMultiSelector

Overview LTableMultiSelector is a concrete implementation of LTableSelector to support a table where multiple cells may be selected. This class supports discontiguous selection.

Methods The methods in this class are:

LTableMultiSelector()	~LTableMultiSelector()
CellIsSelected()	ClickSelect()
DragSelect()	GetFirstSelectedCell()
GetFirstSelectedRow()	InsertCols()
InsertRows()	RemoveCols()
RemoveRows()	SelectAllCells()
SelectCell()	SelectCellBlock()
UnselectAllCells()	UnselectCell()

Data Members The data members in this class are:

mSelectionRgn	mAnchorCell

Operation This class only adds one new method, SelectCellBlock().

Source files (Table Classes)

`LTableMultiSelector.h`

`LTableMultiSelector.cp`

See also LTableSelector

LTableMultiSelector()

Purpose The constructor creates the object.

Access Public

Prototype `LTableMultiSelector( LTableView*inTableView);`

~LTableMultiSelector()

Purpose The destructor destroys the object.

Access Virtual, Public

Prototype `virtual ~LTableMultiSelector();`

CellIsSelected()

Purpose This is an override of CellIsSelected() in LTableSelector.

ClickSelect()

Purpose This is an override of ClickSelect() in LTableSelector.

DragSelect()

Purpose This is an override of DragSelect() in LTableSelector.

GetFirstSelectedCell()

Purpose This is an override of GetFirstSelectedCell() in LTableSelector.

GetFirstSelectedRow()

Purpose This is an override of GetFirstSelectedRow() in LTableSelector.

InsertCols()

Purpose This is an override of InsertCols() in LTableSelector.

InsertRows()

Purpose This is an override of InsertRows() in LTableSelector.

RemoveCols()

Purpose This is an override of RemoveCols() in LTableSelector.

RemoveRows()

Purpose This is an override of RemoveRows() in LTableSelector.

SelectAllCells()

Purpose This is an override of SelectAllCells() in LTableSelector.

SelectCell()

Purpose This is an override of SelectCell() in LTableSelector.

SelectCellBlock()

Purpose Select cells within the specified range, where CellA and CellB are the corners of a rectangular block of cells.

Access Virtual, Protected

Prototype

```
virtual void SelectCellBlock(
     const STableCell&inCellA,
     const STableCell&inCellB );
```

Parameters This method has the following parameters:

`STableCell&`	`inCellA`	The first cell.
`STableCell&`	`inCellB`	The last cell.

Return None

UnselectAllCells()

Purpose This is an override of UnselectAllCells() in LTableSelector.

UnselectCell()

Purpose An override of UnselectCell() in LTableSelector.

mSelectionRgn

Purpose The selection region.

Access Protected

Prototype

```
RgnHandlemSelectionRgn;
```

mAnchorCell

Purpose The anchor cell.

Access Protected

Prototype

```
STableCell mAnchorCell;
```

LTableSelector

Overview LTableSelector is an abstract PowerPlant class that specifies the interface for the selector helper objects. These methods provide behaviors to maintain the selection range in a table.

Methods The methods in this class are:

LTableSelector() ~LTableSelector()
CellIsSelected() ClickSelect()
DragSelect() GetFirstSelectedCell()
GetFirstSelectedRow() InsertCols()
InsertRows() RemoveCols()
RemoveRows() SelectAllCells()
SelectCell() UnselectAllCells()
UnselectCell()

Data Members The data members in this class are:

mTableView

Operation All of the methods in this class are pure virtual or empty.

Source files (Table Classes)

`UTableHelpers.h`

LTableSelector()

Purpose The constructor creates the object.

Access Public

Prototype

```
LTableSelector();
```

Parameters None

~LTableSelector()

Purpose The destructor destroys the object.

Access Virtual, Public

Prototype

```
virtual ~LTableSelector();
```

CellIsSelected()

Purpose Indicates whether a given cell is selected.

Access Pure Virtual, Public

Prototype

```
virtual BooleanCellIsSelected(
     const STableCell&inCell ) const = 0;
```

Parameters This method has the following parameters:

`STableCell&`	`inCell`	A reference to the cell.

Return Returns true if selected, else returns false.

ClickSelect()

Purpose Handle a mouse click within the specified cell.

Access Pure Virtual, Public

Prototype

```
virtual voidClickSelect(
     const STableCell&inCell,
     const SMouseDownEvent&inMouseDown ) = 0;
```

Parameters This method has the following parameters:

`STableCell&`	`inCell`	A reference to the cell.
`SMouseDownEvent&`	`inMouseDown`	The mouse down event info.

Return None

DragSelect()

Purpose Handle a drag operation (the user is pressing the mouse) on a cell.

Access Pure Virtual, Public

Prototype

```
virtual BooleanDragSelect(
     const STableCell&inCell,
     const SMouseDownEvent&inMouseDown ) = 0;
```

Parameters This method has the following parameters:

`STableCell&`	`inCell`	A reference to the cell.
`SMouseDownEvent&`	`inMouseDown`	The mouse down event info.

Return Return true if the mouse never leaves inCell.

GetFirstSelectedCell()

Purpose Get the value of the first selected cell.

Access Pure Virtual, Public

Prototype

```
virtualSTableCellGetFirstSelectedCell() const = 0;
```

Parameters None

Return The cell.

GetFirstSelectedRow()

Purpose Get the first selected row.

Access Pure Virtual, Public

Prototype
```
virtualTableIndexTGetFirstSelectedRow() const = 0;
```

Parameters None

Return The cell.

InsertCols()

Purpose This method handles insertion of columns to the table.

Access Virtual, Public

Prototype
```
virtual voidInsertCols(
     Uint32 inHowMany,
     TableIndexTinAfterCol );
```

Parameters This method has the following parameters:

`Uint32`	`inHowMany`	How many columns to insert.
`TableIndexT`	`inAfterCol`	The column index to insert after.

Return None

InsertRows()

Purpose This method handles insertion of rows to the table.

Access Virtual, Public

Prototype
```
virtual voidInsertRows(
     Uint32 inHowMany,
     TableIndexTinAfterRow );
```

Parameters This method has the following parameters:

`Uint32`	`inHowMany`	How many rows to insert.
`TableIndexT`	`inAfterRow`	The row index to insert after.

Return None

RemoveCols()

Purpose This method handles removal of columns to the table.

Access Virtual, Public

Prototype

```
virtual voidRemoveCols(
     Uint32 inHowMany,
     TableIndexTinFromCol );
```

Parameters This method has the following parameters:

`Uint32`	`inHowMany`	How many columns to insert.
`TableIndexT`	`inFromCol`	The column index to start removal at.

Return None

RemoveRows()

Purpose Remove rows from the table.

Access Virtual, Public

Prototype

```
virtual voidRemoveRows(
     Uint32  inHowMany,
     TableIndexTinFromRow) = 0;
```

Parameters This method has the following parameters:

`Uint32`	`inHowMany`	How many rows to remove.
`TableIndexT`	`inAfterRow`	The row index to start removal at.

Return None

SelectAllCells()

Purpose Selects all cells in the table.

Access Pure Virtual, Public

Prototype

```
virtual voidSelectAllCells() = 0;
```

Parameters None

Return None

SelectCell()

Purpose Select a cell.

Access Pure Virtual, Public

Prototype

```
virtual voidSelectCell(
     const STableCell&inCell ) = 0;
```

Parameters This method has the following parameters:

`STableCell&`	`inCell`	The cell.

Return None

UnselectAllCells()

Purpose Unselect all cells in the table.

Access Pure Virtual, Public

Prototype

```
virtual voidUnselectAllCells() = 0;
```

Parameters None

Return None

UnselectCell()

Purpose Unselect a cell.

Access Pure Virtual, Public

Prototype

```
virtual voidUnselectCell(
     const STableCell&inCell ) = 0;
```

Parameters This method has the following parameters:

`STableCell&`	`inCell`	The cell.

Return None

mTableView

Purpose The table view storage.

Access Protected

Prototype

```
LTableView*mTableView;
```

LTableSingleSelector

Overview LTableSingleSelector is a concrete implementation of LTableSelector for a table that may have one and only one cell selected at a time.

Methods The methods in this class are:

LTableSingleSelector()
~LTableSingleSelector()
CellIsSelected()
ClickSelect()
DragSelect()
GetFirstSelectedCell()
GetFirstSelectedRow()
InsertCols()
InsertRows()
RemoveCols()
RemoveRows()
SelectAllCells()
SelectCell()
UnselectAllCells()
UnselectCell()

Data Members The data members in this class are:

mSelectedCell

Operation This class adds no new methods, but implements every method in LTableSelector.

Source files (Table Classes)

`LTableSingleSelector.h`

`LTableSingleSelector.cp`

See also LTableSelector

LTableSingleSelector()

Purpose The constructor creates the object.

Access Public

Prototype `LTableSingleSelector( LTableView*inTableView);`

~LTableSingleSelector()

Purpose The destructor destroys the object.

Access Virtual, Public

Prototype `virtual ~LTableSingleSelector();`

CellIsSelected()

Purpose This is an override of CellIsSelected() in LTableSelector.

ClickSelect()

Purpose This is an override of ClickSelect() in LTableSelector.

DragSelect()

Purpose This is an override of DragSelect() in LTableSelector.

GetFirstSelectedCell()

Purpose This is an override of GetFirstSelectedCell() in LTableSelector.

GetFirstSelectedRow()

Purpose This is an override of GetFirstSelectedRow() in LTableSelector.

InsertCols()

Purpose This is an override of InsertCols() in LTableSelector.

InsertRows()

Purpose This is an override of InsertRows() in LTableSelector.

RemoveCols()

Purpose This is an override of RemoveCols() in LTableSelector.

RemoveRows()

Purpose This is an override of RemoveRows() in LTableSelector.

SelectAllCells()

Purpose This is an override of SelectAllCells() in LTableSelector.

SelectCell()

Purpose This is an override of SelectCell() in LTableSelector.

UnselectAllCells()

Purpose This is an override of UnselectAllCells() in LTableSelector.

UnselectCell()

Purpose This is an override of UnselectCell() in LTableSelector.

mSelectedCell

Purpose The selected cell.

Access Protected

Prototype

```
STableCell mSelectedCell;
```

LTableStorage

Overview LTableStorage is an abstract PowerPlant class that specifies the interface for the data storage helper objects. These functions provide behaviors to maintain the data associated with a table.

Methods The methods in this class are:

LTableStorage()	~LTableStorage()
FindCellData()	GetCellData()
GetStorageSize()	InsertCols()
InsertRows()	RemoveCols()
RemoveRows()	SetCellData()

Data Members The data member in this class is:

mTableView

Operation All functions in this class are pure virtual, so when you inherit from this class you must provide implementations for every method.

Source files (Table Classes)

`UTableHelpers.h`

LTableStorage()

Purpose The constructor creates an object.

Access Public

Prototype

```
LTableStorage();
```

Parameters None

~LTableStorage()

Purpose The destructor destroys the object.

Access Virtual, Public

Prototype

```
virtual ~LTableStorage();
```

FindCellData()

Purpose Search a cell for the specified data.

Access Pure Virtual, Public

Prototype

```
virtual BooleanFindCellData(
     STableCell&outCell,
     const void*inDataPtr,
     Uint32  inDataSize) const = 0;
```

Parameters This method has the following parameters:

`STableCell&`	`outCell`	A reference to the cell.
`void*`	`inDataPtr`	The buffer pointer.
`Uint32`	`inDataSize`	The buffer size.

Return Return true if the data is found.

GetCellData()

Purpose Retrieve data from a cell.

Access Pure Virtual, Public

Prototype

```
virtual voidGetCellData(
     const STableCell&inCell,
     void    *outDataPtr,
     Uint32  &ioDataSize) const = 0;
```

Parameters This method has the following parameters:

`STableCell&`	`inCell`	A reference to the cell.
`void*`	`outDataPtr`	The buffer pointer.
`Uint32`	`ioDataSize`	The buffer size.

Return None

GetStorageSize()

Purpose Get the storage size.

Access Pure Virtual, Public

Prototype

```
virtual voidGetStorageSize(
     TableIndexT&outRows,
     TableIndexT&outCols ) = 0
```

Parameters This method has the following parameters:

`TableIndexT&`	`outRows`	The number of rows in storage.
`TableIndexT&`	`outCols`	The number of columns in storage.

Return None

InsertCols()

Purpose Insert more columns into storage.

Access Pure Virtual, Public

Prototype

```
virtual voidInsertCols(
     Uint32  inHowMany,
     TableIndexTinAfterCol,
     const void*inDataPtr,
     Uint32  inDataSize) = 0;
```

Parameters This method has the following parameters:

`Uint32`	`inHowMany`	The number of columns to insert.
`TableIndexT`	`inAfterCol`	The column to insert after.
`void*`	`inDataPtr`	The buffer pointer.
`Uint32`	`inDataSize`	The buffer size.

Return None

InsertRows()

Purpose Insert more rows into storage.

Access Pure Virtual, Public

Prototype

```
virtual voidInsertRows(
    Uint32  inHowMany,
    TableIndexTinAfterRow,
    const void*inDataPtr,
    Uint32  inDataSize) = 0;
```

Parameters This method has the following parameters:

`Uint32`	`inHowMany`	The number of columns to insert.
`TableIndexT`	`inAfterRow`	The row to insert after.
`void*`	`inDataPtr`	The buffer pointer.
`Uint32`	`inDataSize`	The buffer size.

Return None

RemoveCols()

Purpose Remove columns from storage.

Access Pure Virtual, Public

Prototype

```
virtual voidRemoveCols(
     Uint32  inHowMany,
     TableIndexTinFromCol ) = 0;
```

Parameters This method has the following parameters:

`Uint32`	`inHowMany`	The number of columns to remove.
`TableIndexT`	`inFromCol`	The column to start removal from.

Return None

RemoveRows()

Purpose Remove rows from storage.

Access Pure Virtual, Public

Prototype

```
virtual voidRemoveRows(
     Uint32  inHowMany,
     TableIndexTinFromRow) = 0;
```

Parameters This method has the following parameters:

`Uint32`	`inHowMany`	The number of rows to remove.
`TableIndexT`	`inFromRow`	The row to start removal from.

Return None

SetCellData()

Purpose Put data into a storage cell.

Access Pure Virtual, Public

Prototype

```
virtual voidSetCellData(
     const STableCell&inCell,
     const void*inDataPtr,
     Uint32  inDataSize ) = 0;
```

Parameters This method has the following parameters:

`STableCell&`	`inCell`	The cell.
`void *`	`inDataPtr`	The buffer pointer.
`Uint32`	`inDataSize`	The buffer size.

Return None

mTableView

Purpose Storage for the pointer to the table.

Access Public

Prototype

```
LTableView *mTableView;
```

LTableView

Overview

LTableView is a PowerPlant class that forms the basis for the rest of the table classes. The class data members store the data you need to create and manage a table. The class member functions implement standard table-related bahavior.

Methods

The methods in this class are:

LTableView()
~LTableView()
ActivateSelf()
AdjustImageSize()
CellIsSelected()
CellToIndex()
ClickCell()
ClickSelect()
ClickSelf()
DeactivateSelf()
DrawCell()
DrawSelf()
FetchIntersectingCells()
FindCellData()
GetCellData()
GetCellHitBy()
GetColWidth()
GetCustomHilite()
GetFirstSelectedCell()
GetHiliteRgn()
GetImageCellBounds()
GetLocalCellRect()
GetNextCell()
GetNextSelectedCell()
GetPreviousCell()
GetPreviousSelectedCell()
GetRowHeight()
GetTableGeometry()
GetTableSelector()
GetTableSize()
GetTableStorage()
HiliteCell()
HiliteCellActively()
HiliteCellInactively()
HiliteSelection()
IndexToCell()
InitTable()
InsertCols()
InsertRows()
IsValidCell()

IsValidCol()	IsValidRow()
PointsAreClose()	RefreshCell()
RefreshCellRange()	RemoveAllCols()
RemoveAllRows()	RemoveCols()
RemoveRows()	ScrollCellIntoFrame()
SelectAllCells()	SelectCell()
SelectionChanged()	SetCellData()
SetColWidth()	SetCustomHilite()
SetDeferAdjustment()	SetRowHeight()
SetTableGeometry()	SetTableSelector()
SetTableStorage()	SetUseDragSelect()
UnselectAllCells()	UnselectCell()

Data Members

The data members in this class are:

mRows	mCols
mTableGeometry	mTableSelector
mTableStorage	mUseDragSelect
mCustomHilite	mDeferAdjustment

Operation

LTableView inherits from LPane and LView. It is thus closely related to the operations of LView. An undertanding of how LPane and LView work is essential to getting the most from this class.

Source files

(Table Classes)

`LTableView.h`

`LTableView.cp`

See also

LAttachable

LView

LPane

LTableView()

Purpose The constructor creates an object.

Access Public

Prototype

```
LTableView();
LTableView( const SPaneInfo&inPaneInfo,
     const SViewInfo&inViewInfo );
LTableView( LStream*inStream );
```

Parameters This method has the following parameters:

`SPaneInfo&`	inPaneInfo	A reference to the pane info.
`SViewInfo&`	inBefore	A reference to the superview.
`LStream*`	inStream	A pointer to the stream object to construct from.

~LTableView()

Purpose The destructor destroys the object.

Access Virtual, Public

Prototype

```
virtual ~LTableView();
```

ActivateSelf()

Purpose Activate a Table

Access Virtual, Protected

Prototype

```
virtual void ActivateSelf();
```

Parameters None

Return None

AdjustImageSize()

Purpose Adjust the Image size of the Table to reflect the number and size of the rows and columns.

Access Virtual, Public

Prototype
```
virtual void AdjustImageSize( BooleaninRefresh );
```

Parameters This method has the following parameters:

Boolean	inRefresh	A boolean indicating whether to refresh or not.

Return None

CellIsSelected()

Purpose Return whether the specified cell is part of the current selection

Access Public

Prototype
```
virtual Boolean CellIsSelected(
     const STableCell&inCell ) const;
```

Parameters This method has the following parameters:

`STableCell&`	inCell	A reference to the cell.

Return Boolean indicating true if the cell is part of the current selection

CellToIndex()

Purpose Pass back the index number for a specified cell

The cell does not have to be in the Table, but the index is zero if the table has no columns or inCell.row is zero.

Cells are ordered by column (across), and then by row (down)

Access Virtual, Public

Prototype

```
virtual void CellToIndex( const STableCell&inCell,
     TableIndexT&outIndex ) const;
```

Parameters This method has the following parameters:

`STableCell&`	inCell	A reference to the cell.
TableIndexT&	outIndex	The index.

ClickCell()

Purpose Handle a mouse click within the specified cell.

Access Virtual, Protected

Prototype

```
virtual void ClickCell(
     const STableCell& inCell,
     const SMouseDownEvent& inMouseDown );
```

Parameters This method has the following parameters:

`STableCell&`	inCell	A reference to the cell.
SMouseDownEvent&	inMouseDown	The mouse down event info.

ClickSelect()

Purpose Adjust selection in response to a click in the specified cell, and return whether or not to process the click as a normal click.

Access Virtual, Public

Prototype

```
virtual Boolean ClickSelect(
```

```
        const STableCell&inCell,
        const SMouseDownEvent&inMouseDown );
```

Parameters This method has the following parameters:

`STableCell&`	inCell	A reference to the cell.
SMouseDownEvent&	inMouseDown	The mouse down event info.

Return Return true if a normal click, false otherwise.

ClickSelf()

Purpose Handle a mouse click within a TableView. This method is an override of ClickSelf().

DeactivateSelf()

Purpose Deactivate a Table. This method is an override of DeactivateSelf().

DrawCell()

Purpose Draw the contents of the specified Cell.

Access Virtual, Protected

Prototype

```
virtual void DrawCell( const STableCell& inCell,
     const Rect& inLocalRect );
```

Parameters This method has the following parameters:

`STableCell&`	`inCell`	A reference to the cell.
`Rect&`	`inLocalRect`	The rectangle coordinates of the cell.

Return None

DrawSelf()

Purpose Draw a TableView. This is an override of DrawSelf() in LPane().

FetchIntersectingCells()

Purpose Pass back the rectangular range of cells, specified by the top left and bottom right cells, that intersects a given rectangle.

Access Virtual, Public

Prototype

```
virtual void FetchIntersectingCells(
     const Rect&inLocalRect,
     STableCell&outTopLeft,
     STableCell&outBotRight) const;
```

Parameters This method has the following parameters:

`STableCell&`	`outTopLeft`	A reference to the top left cell.
`STableCell&`	`outBotRight`	A reference to the bottom right cell.
`Rect&`	`inLocalRect`	The rectangle coordinates of the cell.

Return None

FindCellData()

Purpose Pass back the cell that contains the specified data.

Access Virtual, Public

Prototype

```
virtual Boolean FindCellData(
     STableCell&outCell,
     const void*inDataPtr,
     Uint32  inDataSize) const;
```

Parameters This method has the following parameters:

`STableCell&`	`outCell`	A reference to the cell.
`void*`	`inDataPtr`	The buffer pointer.
`Uint32`	`inDataSize`	The buffer size.

Return Return false if no match is found.

GetCellData()

Purpose Pass back the data for a particular Cell.

Access Virtual, Public

Prototype

```
virtual void GetCellData(
     const STableCell&inCell,
     void    *outDataPtr,
     Uint32  &ioDataSize) const;
```

Parameters This method has the following parameters:

`STableCell&`	`outCell`	A reference to the cell.
`void*`	`outDataPtr`	The buffer pointer. Points to storage which must be allocated by the caller. It may be nil, in which case only the size of the data is passed back.
`Uint32&`	`ioDataSize`	The size of the data buffer (If outDataPtr is nil • input:<ignored> • output:size in bytes of cell's data If outDataPtr is not nil, • input:maximum bytes of data to retrieve • output:actual bytes of data passed back

Return None

GetCellHitBy()

Purpose Pass back the cell which contains the specified point.

If no cell contains the point, return false and

outCell.row = 0 if point is above the Table

outCell.row = mRows + 1 if point is below the Table

outCell.col = 0 if point is to the left of the Table

outCell.col = mCols + 1 if point is to the right of the Table

For example, if the horizontal coordinate is within Column 2, but the vertical coordinate is above the Table, then,

outCell.row = 0

outCell.col = 2

Access Virtual, Public

Prototype

```
virtual Boolean GetCellHitBy(
     const SPoint32&inImagePt,
     STableCell&outCell ) const;
```

Parameters This method has the following parameters:

`SPoint32&`	`inImagePt`	A reference to the point.
`STableCell&`	`outCell`	A reference to the cell.

Return Return false if no cell contains the point.

GetColWidth()

Purpose Return the width of the specified column.

Access Virtual, Public

Prototype
```
virtual Uint16 GetColWidth(
     TableIndexTinCol ) const;
```

Parameters This method has the following parameters:

`TableIndexT`	`inCol`	The column to interrogate.

Return Uint16 indicating the width.

GetCustomHilite()

Purpose Returns the value of the mCustomHilite member.

Access Public

Prototype
```
Boolean GetCustomHilite();
```

Parameters None

GetFirstSelectedCell()

Purpose Return the first selected cell, using the LTableSelector helper object.

Access Public, Virtual

Prototype
```
virtual STableCell GetFirstSelectedCell() const;
```

Parameters None

Return The first selected cell.

GetHiliteRgn()

Purpose Pass back a Region containing the frames of all selected cells which are within the visible rectangle of the Table.

Caller must allocate space for the region.

Access Virtual, Protected

Prototype

```
virtual void GetHiliteRgn(
     RgnHandleioHiliteRgn);
```

Parameters This method has the following parameters:

`RgnHandle`	`ioHiliteRgn`	The region handle.

Return None

GetImageCellBounds()

Purpose Pass back the location in Image coords of the specified Cell.

Access Virtual, Public

Prototype

```
virtual void GetImageCellBounds(
     const STableCell&inCell,
     Int32   &outLeft,
     Int32   &outTop,
     Int32   &outRight,
     Int32   &outBottom) const;
```

Parameters This method has the following parameters:

`STableCell&`	`inCell`	The cell.
`Int32&`	`outLeft`	The left side.
`Int32&`	outTop	The top border.
`Int32&`	`outRight`	The right side.
`Int32&`	`outBottom`	The bottom border.

Return None

GetLocalCellRect()

Purpose Pass back the bounding rectangle of the specified Cell and return whether it intersects the Frame of the TableView.

The bounding rectangle is in Local coordinates so it will always be within QuickDraw space when its within the Frame.

Access Virtual, Public

Prototype

```
virtual Boolean GetLocalCellRect(
     const STableCell&inCell,
     Rect    &outCellRect) const;
```

Parameters This method has the following parameters:

`STableCell&`	`inCell`	The cell.
`Rect&`	`outCellRect`	The rect of the cell.

Return If the bounding rectangle is outside the Frame, return false and set the rectangle to (0,0,0,0).

GetNextCell()

Purpose Pass back the Cell after the specified Cell.

Cells are ordered by column (across), and then by row (down).

Row zero is before the first row. The next cell after row zero and any column is Cell (1,1).

Column zero is before column one. The next cell after row "r" and column zero is Cell (r,1).

Access Virtual, Public

Prototype

```
virtual Boolean GetNextCell(
     STableCell&ioCell) const;
```

Parameters This method has the following parameters:

`STableCell&`	`ioCell`	The cell.

Return Return false if there is no cell after the specified one, andpass back Cell (0,0). Otherwise, return true and pass back the next Cell's indexes.

GetNextSelectedCell()

Purpose Pass back the selected Cell after the specified Cell.

This function uses the same ordering rules as GetNextCell(). Pass in Cell (0,0) to find the first selected Cell.

Access Virtual, Public

Prototype

```
virtual Boolean GetNextSelectedCell(
     STableCell&ioCell ) const;
```

Parameters This method has the following parameters:

`STableCell&`	`ioCell`	The cell.

Return Return false if there is no selected Cell after the specified one.

GetPreviousCell()

Purpose Pass back the Cell before the specified Cell.

Cells are ordered by column (across), and then by row (down).

Cell (mRows,mCols) is the one before Cell (0,c), where c is any column. Thus, you can pass in Cell (0,0) to get the last cell.

Cell (mRows,mCols) is also the cell before any Cell that is beyond the limits of the table.

Column zero is before column one. The cell before row "r" and column zero is Cell (r-1,mCols).

Access Virtual, Public

Prototype

```
virtual Boolean GetPreviousCell(
     STableCell&ioCell ) const;
```

Parameters This method has the following parameters:

`STableCell&`	`ioCell`	The cell.

Return Return false if there is no cell before the specified one, and pass back Cell (0,0). Otherwise, return true and pass back the previous Cell's indexes.

GetPreviousSelectedCell()

Purpose Pass back the selected Cell before the specified Cell.

This function uses the same ordering rules as GetPreviousCell(). Pass in Cell (0,0) to find the last selected Cell.

Access Virtual, Public

Prototype

```
virtual Boolean GetPreviousSelectedCell(
     STableCell&ioCell ) const;
```

Parameters This method has the following parameters:

`STableCell&`	`ioCell`	The cell.

Return Return false if there is no selected cell before the cell indicated, and pass back Cell (0,0). Otherwise, return true and pass back the previous Cell's indexes.

GetRowHeight()

Purpose Return the height of the specified row.

Access Virtual, Public

Prototype

```
virtual Uint16 GetRowHeight(
     TableIndexTinRow ) const;
```

Parameters This method has the following parameters:

`TableIndexT`	`inRow`	The table index.

Return A Uint16 indicating the height of the row i display coordinates.

GetTableGeometry()

Purpose Return the value of the mTableGeometry data member.

Access Public

Prototype

```
LTableGeometry* GetTableGeometry();
```

Parameters None

Return A pointer to an LTableGeometry object.

GetTableSelector()

Purpose Return the value of the mTableSelector data member.

Access Public

Prototype

```
LTableSelector* GetTableSelector();
```

Parameters None

Return A pointer to an LTableSelector object.

GetTableSize()

Purpose Pass back the number of rows and columns in a TableView.

Access Public

Prototype

```
void GetTableSize(
     TableIndexT&outRows,
     TableIndexT&outCols ) const;
```

Parameters This method has the following parameters:

`TableIndexT&`	`outRows`	The number of rows.
`TableIndexT&`	`outCols`	The number of columns.

Return None

GetTableStorage()

Purpose Returns the value of the mTableStorage data member.

Access Public

Prototype

```
LTableStorage* GetTableStorage();
```

Parameters None

Return A pointer to an `LTableStorage` object.

HiliteCell()

Purpose Draw or undraw hiliting for the specified Cell

Access Virtual, Public

Prototype

```
virtual void HiliteCell(
     const STableCell&inCell,
     Boolean inHilite );
```

Parameters This method has the following parameters:

`STableCell&`	`inCell`	The cell.
`Boolean`	`inHilite`	Whether to hilite or not.

Return None

HiliteCellActively()

Purpose Draw or undraw active hiliting for a Cell.

Access Virtual, Protected

Prototype

```
virtual void HiliteCellActively(
     const STableCell&inCell,
     Boolean  inHilite );
```

Parameters This method has the following parameters:

`STableCell&`	`inCell`	The cell.
`Boolean`	`inHilite`	Whether to hilite or not.

Return None

HiliteCellInactively()

Purpose Draw or undraw inactive hiliting for a Cell

Access Virtual, Protected

Prototype

```
virtual void HiliteCellInactively(
     const STableCell&inCell,
     Boolean  inHilite);
```

Parameters This method has the following parameters:

`STableCell&`	`inCell`	The cell.
`Boolean`	`inHilite`	Whether to hilite or not.

Return None

HiliteSelection()

Purpose Draw or undraw hiliting for the current selection in either the active or inactive state

Access Virtual, Public

Prototype

```
virtual void HiliteSelection(
     BooleaninActively,
     BooleaninHilite );
```

Parameters This method has the following parameters:

`Boolean`	`inActively`	Whether actively or not.
`Boolean`	`inHilite`	Whether to hilite or not.

Return None

IndexToCell()

Purpose Pass back the cell for a specified index number.

Index number does not have to refer to an actual Cell, but Cell is (0,0) if Table has no columns or inIndex is zero.

Access Virtual, Public

Prototype

```
virtual void IndexToCell(
     TableIndexTinIndex,
     STableCell&outCell ) const;
```

Parameters This method has the following parameters:

`TableIndexT`	`inIndex`	The table index.
`STableCell&`	`outCell`	The cell.

Return None

InitTable()

Purpose Private Initializer

Access Private

Prototype
```
void InitTable();
```

Parameters None

Return None

InsertCols()

Purpose Add columns to a TableView.

Use inAfterCol of 0 to insert columns at the beginning All cells in the newly inserted rows have the same data.

Access Virtual, Public

Prototype
```
virtual void InsertCols(
     Uint32inHowMany,
     TableIndexTinAfterCol,
     const void*inDataPtr,
     Uint32inDataSize,
     BooleaninRefresh);
```

Parameters This method has the following parameters:

`Uint32`	`inHowMany`	How many columns to add.
`TableIndexT`	`inAfterCol`	The column to insert after.
`void*`	`inDataPtr`	The buffer pointer.
`Uint32`	`inDataSize`	The size of the buffer.
`Boolean`	`inRefresh`	Whether to refresh or not.

Return None

InsertRows()

Purpose Add rows to a TableView.

Use inAfterRow of 0 to insert rows at the beginning All cells in the newly inserted rows have the same data.

Access Virtual, Public

Prototype

```
virtual void InsertRows(
     Uint32inHowMany,
     TableIndexTinAfterRow,
     const void*inDataPtr,
     Uint32inDataSize,
     BooleaninRefresh);
```

Parameters This method has the following parameters:

`Uint32`	`inHowMany`	How many columns to add.
`TableIndexT`	`inAfterRow`	The row to insert after.
`void*`	`inDataPtr`	The buffer pointer.
`Uint32`	`inDataSize`	The size of the buffer.
`Boolean`	`inRefresh`	Whether to refresh or not.

Return None

IsValidCell()

Purpose Return whether a TableView includes a specified Cell.

Access Public

Prototype

```
Boolean IsValidCell(
     const STableCell&inCell ) const;
```

Parameters This method has the following parameters:

`STableCell&`	`inCell`	The cell index to check.

Return Boolean indicating whether the cell is valid or not.

IsValidCol()

Purpose Return whether a TableView includes a specified column.

Access Public

Prototype

```
Boolean IsValidCol(
     TableIndexTinCol ) const;
```

Parameters This method has the following parameters:

`STableCell&`	`inCol`	The cell column to check.

Return Boolean indicating whether the cell includes the column or not.

IsValidRow()

Purpose Return whether a TableView includes a specified row

Access Public

Prototype

```
Boolean IsValidRow(
     TableIndexTinRow ) const;
```

Parameters This method has the following parameters:

`TableIndexT`	`inRow`	The cell row to check.

Return Boolean indicating whether the cell includes the row or not.

PointsAreClose()

Purpose Indicate whether the two points are close enough to be part of a multi-click. Points are in Local coordinates.

Points must meet the standard test (inherited function) as well as be inside the same cell.

Access Virtual, Public

Prototype

```
virtual Boolean PointsAreClose(
     PointinFirstPt,
     PointinSecondPt ) const;
```

Parameters This method has the following parameters:

`Point`	`inFirstPt`	The first point.
Point	`inSecondPt`	The second point.

Return Return true if the two points are close enough, else return false.

RefreshCell()

Purpose Invalidate the area occupied by the specified cell so that its contents will be redrawn during the next update event

Access Virtual, Public

Prototype

```
virtual void RefreshCell(
     const STableCell&inCell );
```

Parameters This method has the following parameters:

`STableCell&`	`inCell`	The cell.

Return None

RefreshCellRange()

Purpose Invalidate a rectangular block of cells so that their contents will be redrawn during the next update event.

Access Virtual, Public

Prototype

```
virtual void RefreshCellRange(
     const STableCell&inTopLeft,
     const STableCell&inBotRight );
```

Parameters This method has the following parameters:

`STableCell&`	`inTopLeft`	The top left cell.
`STableCell&`	`inBotRight`	The bottom right cell.

Return None

RemoveAllCols()

Purpose Remove all columns in a table.

Access Virtual, Public

Prototype

```
virtual void RemoveAllCols(
     BooleaninRefresh );
```

Parameters This method has the following parameters:

`Boolean`	`inRefresh`	Whether to refresh or not.

Return None

RemoveAllRows()

Purpose Remove all rows from a table.

Access Virtual, Public

Prototype

```
virtual void RemoveAllRows(
     BooleaninRefresh );
```

Parameters This method has the following parameters:

`Boolean`	`inRefresh`	Whether to refresh or not.

Return None

RemoveCols()

Purpose Delete columns from a TableView

Access Virtual, Public

Prototype

```
virtual void RemoveCols(
     Uint32inHowMany,
     TableIndexTinFromCol,
     BooleaninRefresh );
```

Parameters This method has the following parameters:

`Uint32`	`inHowMany`	How many columns to delete.
`TableIndexT`	`inFromCol`	The column to start from.
`Boolean`	`inRefresh`	Whether or not to refresh.

Return None

RemoveRows()

Purpose Delete rows from a TableView

Access Virtual, Public

Prototype

```
virtual void RemoveRows(
     Uint32inHowMany,
     TableIndexTinFromRow,
     BooleaninRefresh );
```

Parameters This method has the following parameters:

`Uint32`	`inHowMany`	How many rows to delete.
`TableIndexT`	`inFromRow`	The row to start from.
`Boolean`	`inRefresh`	Whether or not to refresh.

Return None

ScrollCellIntoFrame()

Purpose Scroll the TableView as little as possible to move the specified Cell so that it's entirely within the Frame of the TableView.

If Cell is wider and/or taller than Frame, align Cell to left/top of Frame.

Access Virtual, Public

Prototype

```
virtual void ScrollCellIntoFrame(
     const STableCell&inCell );
```

Parameters This method has the following parameters:

`STableCell&`	`inCell`	The cell.

Return None

SelectAllCells()

Purpose Select all Cells in a Table

Access VIrtual, Public

Prototype

```
virtual void SelectAllCells();
```

Parameters None

Return None

SelectCell()

Purpose Add the specified cell to the current selection.

Access Virtual, Public

Prototype

```
virtual void SelectCell(
     const STableCell&inCell );
```

Parameters This method has the following parameters:

`STableCell&`	`inCell`	The cell.

Return None

SelectionChanged()

Purpose Notification that the cells which are selected has changed

Access Virtual, Public

Prototype

```
virtual void SelectionChanged();
```

Parameters None

Return None

SetCellData()

Purpose Specify the data associated with a particular Cell

Access Virtual, Public

Prototype

```
virtual void SetCellData(
     const STableCell&inCell,
     const void*inDataPtr,
     Uint32  inDataSize );
```

Parameters This method has the following parameters:

`STableCell&`	`inCell`	The cell.
`void*`	`inDataPtr`	The buffer pointer.
`Uint32`	`inDataSize`	The buffer size.

Return None

SetColWidth()

Purpose Set the width of the specified columns.

Access Virtual, Public

Prototype

```
virtual void SetColWidth(
     Uint16inWidth,
     TableIndexTinFromCol,
     TableIndexTinToCol);
```

Parameters This method has the following parameters:

`Uint16`	`inWidth`	The cell width.
`TableIndexT`	`inFromCol`	The starting column.
`TableIndexT`	`inToCol`	The ending column.

Return None

SetCustomHilite()

Purpose Set tha value of the mCustomHilite data member.

Access Public

Prototype

```
void SetCustomHilite( Boolean inCustom );
```

Parameters This method has the following parameters:

`Boolean`	`inCustom`	The value to set.

Return None

SetDeferAdjustment()

Purpose Specify whether to defer adjustment of Table to account for a change in the size of the Table (number or size of rows or columns).

Calling AdjustImageSize() can take a lot of time, especially for large Tables and those with variable row or column sizes. If you are going to make a series of changes to a Table, there's no need to adjust the Image size until after all those changes are made.

Therefore, call SetDeferAdjustment(true) before making your changes, then call SetDeferAdjustment(false) afterwards. Or use the StDeferTableAdjustment stack-based class.

Access Public

Prototype

```
void SetDeferAdjustment( BooleaninDefer );
```

Parameters This method has the following parameters:

`Boolean`	`inDefer`	Whether to defer or not.

Return None

SetRowHeight()

Purpose Set the height of the specified rows.

Access Virtual, Public

Prototype

```
virtual void SetRowHeight( Uint16inHeight,
     TableIndexTinFromRow,
     TableIndexTinToRow );
```

Parameters This method has the following parameters:

`Uint16`	`inHeight`	The height to use.
`TableIndexT`	`inFromRow`	The starting row.
`TableIndexT`	`inToRow`	The ending row.

Return None

SetTableGeometry()

Purpose Specify the Geometry for a TableView.

The Geometry determines the dimensions of the Cells in the Table

Access Virtual, Public

Prototype

```
virtual void SetTableGeometry(
     LTableGeometry*inTableGeometry );
```

Parameters This method has the following parameters:

`LTableGeometry*`	`inTableGeometry`	The table geometry object.

Return None

SetTableSelector()

Purpose Specify the Selector for a TableView. The Selector stores and controls which Cells in a Table are selected. This sets the value of mTableSelector.

Access Virtual, Public

Prototype

```
virtual void SetTableSelector(
     LTableSelector*inTableSelector );
```

Parameters This method has the following parameters:

`LTableSelector*`	`inTableSelector`	The table selector pointer.

Return None

SetTableStorage()

Purpose Specify the Storage for a TableView. The Storage holds the data for each Cell in a Table. This sets the value of mTableStorage.

Access Virtual, Public

Prototype

```
virtual void SetTableStorage(
    LTableStorage*inTableStorage );
```

Parameters This method has the following parameters:

`LTableStorage*`	`inTableStorage`	The table storage pointer.

Return None

SetUseDragSelect()

Purpose Indicate whether to use drag select or not. This sets the value of the mUseDragSelect data member.

Access Public

Prototype

```
void SetUseDragSelect( Boolean inUseIt);
```

Parameters This method has the following parameters:

`Boolean`	`inUseIt`	The value to set.

Return None

UnselectAllCells()

Purpose Unselect all currently selected cells so there is no selection

Access Virtual, Public

Prototype
```
virtual void UnselectAllCells();
```

Parameters None

Return None

UnselectCell()

Purpose Remove the specified cell from the current selection

Access Virtual, Public

Prototype
```
virtual void UnselectCell(const STableCell&inCell);
```

Parameters This method has the following parameters:

`STableCell&`	`inCell`	The cell.

Return None

mRows

Purpose Data member that contains number of rows.

Access Protected

Prototype
```
TableIndexT mRows;
```

mCols

Purpose Data member that contains number of columns.

Access Protected

Prototype `TableIndexT mCols;`

mTableGeometry

Purpose Storage for the table geometry.

Access Protected

Prototype `LTableGeometry *mTableGeometry;`

mTableSelector

Purpose Storage for the table selector

Access Protected

Prototype `LTableSelector *mTableSelector;`

mTableStorage

Purpose Table storage.

Access Protected

Prototype `LTableStorage *mTableStorage;`

mUseDragSelect

Purpose Storage for whether or not to use drag select.

Access Protected

Prototype `Boolean mUseDragSelect;`

mCustomHilite

Purpose Whether to use custom hilite or not.

Access Protected

Prototype

```
Boolean mCustomHilite;
```

mDeferAdjustment

Purpose Whether to defer adjustments to the table or not.

Access Protected

Prototype

```
Boolean mDeferAdjustment;
```

LTCPEndpoint

Overview

`LTCPEndpoint` is a PowerPlant class that is used for implementing Internet TCP endpoints. TCP/IP is the session-oriented protocol of the Internet, and is the layer upon which most of the familiar Internet protocols (HTTP, FTP, SMTP, etc.) are built.

Methods

The methods in this class are:

LTCPEndpoint()	~LTCPEndpoint()
AbortiveDisconnect()	AcceptIncoming()
AcceptRemoteDisconnect()	Connect()
Disconnect()	GetAmountUnread()
GetRemoteHostAddress()	Listen()
Receive()	ReceiveChar()
ReceiveData()	ReceiveDataUntilMatch()
ReceiveLine()	RejectIncoming()
Send()	SendCStr()
SendData()	SendDisconnect()
SendHandle()	SendPStr()
SendPtr()	

Data Members

There are no data members in this class.

Operation

In PowerPlant, there are two subclasses of LTCPEndpoint, named LMacTCPTCPEndpoint and LOpenTptTCPEndpoint. The appropriate class is created automatically when you call UNetworkFactory::CreateTCPEndpoint().

Note that most of these methods are pure virtual, requiring you to implement them if you inherit from this class.

Source files

(Networking Classes)

`LTCPEndpoint.h`

`LTCPEndpoint.cp`

See also LEndpoint

LMacTCPTCPEndpoint

LOpenTptTCPEndpoint

LTCPEndpoint()

Purpose The constructor creates the object.

Access Public

Prototype
```
LTCPEndpoint();
```

Parameters None

~LTCPEndpoint()

Purpose The destructor destroys the `LTCPEndpoint` object.

Access Virtual, Public

Prototype
```
virtual ~LTCPEndpoint();
```

AbortiveDisconnect()

Purpose Called for a disconnect on abort. You must override this and provide functionality if you inherit from this class.

Access Pure virtual, Public

Prototype
```
virtual void AbortiveDisconnect() = 0;
```

Parameters None

Return None

AcceptIncoming()

Purpose Call to accept an incoming connection.

Access Pure virtual, Public

Prototype
```
virtual void AcceptIncoming(
     LTCPEndpoint* inEndpoint ) = 0;
```

Parameters The parameters for this method are:

`LTCPEndpoint*`	`inEndpoint`	The endpoint.

Return None

AcceptRemoteDisconnect()

Purpose Called for accepting a remote disconnect.

Access Pure virtual, Public

Prototype
```
virtual void AcceptRemoteDisconnect() = 0;
```

Parameters None

Return None

Connect()

Purpose Called to connect.

Access Pure virtual, Public

Prototype
```
virtual void Connect(
     LInternetAddress& inRemoteAddress,
     Uint32 inTimeoutSeconds = Timeout_None ) = 0;
```

Parameters The parameters for this method are:

`LInternetAddress&`	`inRemoteAddress`	The address.
`Uint32`	`inTimeoutSeconds`	The timeout value, default is `Timeout_None`.

Return None

Disconnect()

Purpose Called to disconnect.

Access Pure virtual, Public

Prototype
```
virtual void Disconnect() = 0;
```

Parameters None

Return None

GetAmountUnread()

Purpose Learn how much unread data remains.

Access Pure virtual, Public

Prototype
```
virtual Uint32 GetAmountUnread() = 0;
```

Parameters None

Return Returns the amount of unread data.

GetRemoteHostAddress()

Purpose Retrieve the address of the remote host.

Access Pure virtual, Public

Prototype
```
virtual LInternetAddress *GetRemoteHostAddress() =
0;
```

Parameters None

Return A pointer to the `LInternetAddress`.

Listen()

Purpose Listen for data.

Access Pure virtual, Public

Prototype
```
virtual void Listen() = 0;
```

Parameters None

Return None

Receive()

Purpose Receive data.

Access Pure virtual, Public

Prototype
```
virtual void Receive( void* outDataBuffer, Uint32&
ioDataSize );
```

Parameters The parameters for this method are:

`void*`	`outDataBuffer`	The buffer pointer.
`Uint32&`	`ioDataSize`	The data size.

Return None

ReceiveChar()

Purpose Receive a character.

Access Virtual, Public

Prototype

```
virtual Boolean ReceiveChar( char& outChar, Uint32
     inTimeoutSeconds = Timeout_None);
```

Parameters The parameters for this method are:

`char&`	`outChar`	The buffer pointer.
`Uint32`	`inTimeoutSeconds`	The timeout, the default is Timeout_None.

Return Return true if a character is received.

ReceiveData()

Purpose Receive data over a connection.

Access Pure virtual, Public

Prototype

```
virtual void ReceiveData( void* outDataBuffer,
     Uint32& ioDataSize,
     Boolean& outExpedited,
     Uint32 inTimeoutSeconds = Timeout_None) = 0;
```

Parameters The parameters for this method are:

`void*`	`outDataBuffer`	The buffer pointer.
`Uint32`	`ioDataSize`	The data size.

`Boolean&`	`outExpedited`	Whether expedited or not.
`Uint32`	`inTimeoutSeconds`	The timeout, the default is `Timeout_None`.

Return None

ReceiveDataUntilMatch()

Purpose Receive data until a specified character is found.

Access Virtual, Public

Prototype

```
virtual Boolean ReceiveDataUntilMatch(
     void* outDataBuffer,
     Uint32& ioDataSize,
     Boolean& outExpedited,
     Uint32 inTimeoutSeconds,
     char inMatchChar = 0x0D );
```

Parameters The parameters for this method are:

`void*`	`outDataBuffer`	The buffer pointer.
`Uint32`	`ioDataSize`	The data size.
`Boolean&`	`outExpedited`	Whether expedited or not.
`Uint32`	`inTimeoutSeconds`	The timeout, the default is `Timeout_None`.
`char`	`inMatchChar`	The character to search for, the default is 0x0D.

Return Returns true if the specified character is found before the timeout, else returns false.

ReceiveLine()

Purpose Receive a line of data.

Access Virtual, Public

Prototype

```
virtual Boolean ReceiveLine(
     char * outString,
     Uint32& ioDataSize,
     Uint32 inTimeoutSeconds,
     Boolean inUseLF = false );
```

Parameters The parameters for this method are:

`char*`	`outString`	The buffer pointer to fill.
`Uint32&`	`ioDataSize`	The data size.
`Uint32`	`inTimeoutSeconds`	The timeout, the default is `Timeout_None`.
`Boolean`	`inUseLF`	Whether to use a line feed or not, the default is false. If true then we look for a LF instead of CR to terminate the line.

Return Returns the value of ReceiveDataUntilMatch().

RejectIncoming()

Purpose Call to reject incoming data.

Access Pure virtual, Public

Prototype

```
void RejectIncoming() = 0;
```

Parameters None

Return None

Send()

Purpose Calls SendData() to send data.

Access Virtual, Public

Prototype

```
virtual void Send( void* inData,
     Uint32 inDataSize );
```

Parameters The parameters for this method are:

`void*`	`inData`	The buffer pointer to send.
`Uint32&`	`inDataSize`	The data size.

Return None

SendCStr()

Purpose Utility method for SendData().

Access Virtual, Public

Prototype

```
virtual void SendCStr(char* inString);
```

Parameters The parameters for this method are:

`char*`	`inString`	The C string to send.

Return None

SendData()

Purpose A routine to actually send the data out.

Access Pure virtual, Public

Prototype

```
virtual void SendData( void* inData,
     Uint32 inDataSize,
     Boolean inExpedited = false,
     Uint32 inTimeoutSeconds = Timeout_None ) = 0;
```

Parameters The parameters for this method are:

`void*`	`inData`	The data to send.
`Uint32`	`inDataSize`	The data size.
`Boolean`	`inExpedited`	Whether to expedite or not, default is false.
`Uint32`	`inTimeoutSeconds`	The timeout, default is `Timeout_None`.

Return None

SendDisconnect()

Purpose Send a disconnect.

Access Pure virtual, Public

Prototype

```
virtual void SendDisconnect() = 0;
```

Parameters None

Return None

SendHandle()

Purpose Utility method for SendData().

Access Virtual, Public

Prototype

```
virtual void SendHandle(Handle inHandle);
```

Parameters The parameter for this method is:

`Handle`	`inHandle`	The handle to send.

Return None

SendPStr()

Purpose Utility method for SendData().

Access Virtual, Public

Prototype

```
virtual void SendPStr(ConstStringPtr inString);
```

Parameters The parameter for this method is:

`ConstStringPtr`	`inString`	The Pascal string to send.

Return None

SendPtr()

Purpose Utility method for SendData().

Access Virtual, Public

Prototype

```
virtual void SendPtr(Ptr inPtr);
```

Parameters The parameter for this method is:

`ConstStringPtr`	`inString`	The data to send.

Return None

LTEClearAction

Overview LTEClearAction is a PowerPlant class that is used for handling the clear action for a TextEdit field.

Methods The methods in this class are:

LTEClearAction() ~LTEClearAction()

RedoSelf()

Data Members There are no data members in this class.

Operation This is a simple class for handling the clearing of the TextEdit field.

Source files (Action Classes)

`UTETextAction.h`

`UTETextAction.cp`

Ancestors LAction

LTETextAction

LTEClearAction()

Purpose The constructor creates the object.

Access Public

Prototype

```
LTEClearAction();
```

Parameters None

~LTEClearAction()

Purpose The destructor destroys the object.

Access Virtual, Public

Prototype

```
virtual ~LTEClearAction();
```

RedoSelf()

Purpose This is an override of RedoSelf() in LAction.

LTECutAction

Overview LTECutAction is a PowerPlant class that is used for handling the cut action for a TextEdit field.

Methods The methods in this class are:

LTECutAction() ~LTECutAction()

RedoSelf()

Data Members There are no data members in this class.

Operation This is a simple class for handling cuts from the TextEdit field.

Source files (Action Classes)

`UTETextAction.h`

`UTETextAction.cp`

Ancestors LAction

LTETextAction

LTECutAction()

Purpose The constructor creates an object from the passed-in parameters.

Access Public

Prototype

```
LTECutAction( TEHandle inMacTEH,
     LCommander *inTextCommander,
     LPane *inTextPane );
```

Parameters The parameters for this constructor are:

`TEHandle`	`inMacTEH`	The Mac OS TextEdit handle.
`LCommander*`	`inTextCommander`	The pointer to the text Commander.
`LPane*`	`inTextPane`	The pointer to the Pane.

~LTECutAction()

Purpose The destructor destroys the object.

Access Virtual, Public

Prototype

```
virtual ~LTECutAction();
```

RedoSelf()

Purpose This is an override of RedoSelf() in LAction.

LTEPasteAction

Overview LTEPasteAction is a PowerPlant class that is used for handling the paste action for a TextEdit field.

Methods The methods in this class are:

LTEPasteAction() ~LTEPasteAction()

RedoSelf() UndoSelf()

Data Members The data members in this class are:

mPastedTextH

Operation This is a simple class for handling pastes from the TextEdit field.

Source files (Action Classes)

`UTETextAction.h`

`UTETextAction.cp`

Ancestors LAction

LTETextAction

LTEPasteAction()

Purpose The constructor creates an object from the passed-in parameters.

Access Public

Prototype

```
LTEPasteAction( TEHandle inMacTEH,
     LCommander *inTextCommander,
     LPane *inTextPane );
```

Parameters The parameters for this constructor are:

`TEHandle`	`inMacTEH`	The Mac OS TextEdit handle.
`LCommander*`	`inTextCommander`	The pointer to the text Commander.
`LPane*`	`inTextPane`	The pointer to the Pane.

~LTEPasteAction()

Purpose The destructor destroys the object.

Access Virtual, Public

Prototype

```
virtual ~LTEPasteAction();
```

RedoSelf()

Purpose This is an override of RedoSelf() in LAction.

UndoSelf()

Purpose This is an override of UndoSelf() in LAction.

mPastedTextH

Purpose The handle to the pasted text.

Access Protected

Prototype

```
Handle mPastedTextH;
```

LTETextAction

Overview LTETextAction is a PowerPlant class that is used for handling the common editing actions for a TextEdit field.

Methods The methods in this class are:

LTETextAction()	~LTETextAction()
CanRedo()	CanUndo()
Redo()	Undo()
UndoSelf()	

Data Members The data members in this class are:

mTextCommander	mTextPane
mMacTEH	mActionCommand
mDeletedTextH	mDeletedTextLen
mSelStart	mSelEnd

Operation This is a simple class for handling operations for the TextEdit field.

Source files (Action Classes)

`UTETextAction.h`

`UTETextAction.cp`

Ancestors LAction

LTETextAction

LTETextAction()

Purpose The constructor creates an object from the passed-in parameters.

Access Public

Prototype

```
LTETextAction( Int16 inDescriptionIndex,
     CommandT inActionCommand,
     TEHandle inMacTEH,
     LCommander *inTextCommander,
     LPane *inTextPane,
     Boolean inAlreadyDone );
```

Parameters The parameters for this constructor are:

`Int16`	`inDescriptionIndex`	The description index to send to the LAction() constructor.
`CommandT`	`inActionCommand`	The action command.
`TEHandle`	`inMacTEH`	The Mac OS TextEdit handle.
`LCommander*`	`inTextCommander`	The commander.
`LPane*`	`inTextPane`	The pane for the text.
`Boolean`	`inAlreadyDone`	This is passed to the LAction() constructor.

~LTETextAction()

Purpose The destructor destroys the object.

Access Virtual, Public

Prototype

```
virtual ~LTETextAction();
```

CanRedo()

Purpose This is an override of CanRedo() in LAction.

CanUndo()

Purpose This is an override of CanUndo() in LAction.

Redo()

Purpose This is an override of Redo() in LAction.

Undo()

Purpose This is an override of Undo() in LAction.

UndoSelf()

Purpose This is an override of UndoSelf() in LAction.

mTextCommander

Purpose Pointer storage for the text Commander.

Access Protected

Prototype
```
LCommander *mTextCommander;
```

mTextPane

Purpose A pointer to the text Pane.

Access Protected

Prototype
```
LPane *mTextPane;
```

mMacTEH

Purpose The Mac OS TextEdit handle.

Access Protected

Prototype `TEHandle mMacTEH;`

mActionCommand

Purpose Storage for the action command.

Access Protected

Prototype `MessageT mActionCommand;`

mDeletedTextH

Purpose Storage for the handle to deleted text.

Access Protected

Prototype `Handle mDeletedTextH;`

mDeletedTextLen

Purpose Storage for the length of the deleted text.

Access Protected

Prototype `Int32 mDeletedTextLen;`

mSelStart

Purpose The start of the selection.

Access Protected

Prototype `Int16 mSelStart;`

mSelEnd

Purpose The end of the selection.

Access Protected

Prototype

```
Int16 mSelEnd;
```

LTETypingAction

Overview LTETypingAction is a PowerPlant class that is used for handling common typing actions.

Methods The methods in this class are:

LTETypingAction()	~LTETypingAction()
BackwardErase()	ForwardErase()
InputCharacter()	RedoSelf()
Reset()	UndoSelf()

Data Members The data members in this class are:

mTypedTextH	mTypingStart
mTypingEnd	

Operation This is a simple class for handling typing in the TextEdit field.

Source files (Action Classes)

`UTETextAction.h`

`UTETextAction.cp`

Ancestors LAction

LTETextAction

LTETypingAction()

Purpose The constructor creates the object from the passed-in parameters.

Access Public

Prototype

```
LTETypingAction( TEHandle inMacTEH,
     LCommander *inTextCommander,
     LPane *inTextPane );
```

Parameters The parameters for this constructor are:

`TEHandle`	`inMacTEH`	The Mac OS TextEdit handle.
`LCommander*`	`inTextCommander`	The pointer to the text Commander.
`LPane*`	`inTextPane`	The pointer to the Pane.

~LTETypingAction()

Purpose The destructor destroys the object.

Access Virtual, Public

Prototype

```
virtual ~LTETypingAction();
```

BackwardErase()

Purpose Handle the backward delete typing action. Backward delete erases the current selection if one or more characters are selected. If the selection is a single insertion point, then backward delete erases the one character before the insertion point

Access Virtual, Public

Prototype

```
virtual void BackwardErase();
```

Parameters None

Return None

ForwardErase()

Purpose Handle forward delete typing action. Forward delete erases the current selection if one or more characters are selected. If the selection is a single insertion point, then forward delete erases the one character after the insertion point.

Access Virtual, Public

Prototype
```
virtual void ForwardErase();
```

Parameters None

Return None

InputCharacter()

Purpose Handle an input character typing action.

Access Virtual, Public

Prototype
```
virtual void InputCharacter( Int16 inChar );
```

Parameters The parameters for this constructor are:

`Int16`	`inChar`	The character that was typed.

Return None

RedoSelf()

Purpose Redo a TypingAction by restoring the last typing sequence. This is an override of RedoSelf() in LAction.

Reset()

Purpose Re-initialize the state of the typing action.

Access Virtual, Public

Prototype `virtual void Reset();`

Parameters None

Return None

UndoSelf()

Purpose Undo a typing action by restoring the text and selection that existed before the current typing sequence started. This is an override of UndoSelf() in LAction.

mTypedTextH

Purpose The handle to the typed text.

Access Protected

Prototype `Handle mTypedTextH;`

mTypingStart

Purpose The start of the typing.

Access Protected

Prototype `Int16 mTypingStart;`

mTypingEnd

Purpose The end of the typing.

Access Protected

Prototype `Int16 mTypingEnd;`

LTextButton

Overview

LTextButton is a PowerPlant class that is used for managing buttons that have text instead of graphics in them. There is no standard Mac OS item that matches this class.

Methods

The methods in this class are:

LTextButton()	~LTextButton()
DrawSelf()	GetDescriptor()
HotSpotAction()	HotSpotResult()
SetDescriptor()	SetValue()

Data Members

The data members in this class are:

mText	mTextTraitsID
mSelectedStyle	

Operation

Clicking an LTextButton toggles the button's state between on and off, like a radio button. The different state is represented visually by a change in the text style. The button style may become underline, bold, italic, outline, shadow, condensed, extended, or any combination of these style options.

When the user clicks the button, a message is sent to all the button's Listeners. In a typical scenario, you want some action to occur immediately when the user clicks a text button. You should specify a value message that uniquely identifies the button to any Listener.

Source files

(Pane Classes)

`LTextButton.h`

`LTextButton.cp`

Ancestors The ancestors for this class are:

LAttachable LControl
LBroadcaster LPane

LTextButton()

Purpose The constructors create objects from the passed-in parameters.

Access Public

Prototype

```
LTextButton();
LTextButton( LStream* inStream );
```

Parameters The stream constructor has the following parameter:

`LStream*`	`inStream`	A pointer to a stream object that contains the information to create the LTextButton object.

~LTextButton()

Purpose The destructor destroys the object.

Access Virtual, Public

Prototype

```
virtual ~LTextButton();
```

DrawSelf()

Purpose Draw this text in with appropriate text style for the selected status. This is an override of DrawSelf() in LPane.

GetDescriptor()

Purpose Return the text of the button. This is an override of GetDescriptor() in LPane.

HotSpotAction()

Purpose Toggle between highlighted and plain states, depending on mouse location. This is an override of HotSpotAction() in LControl.

HotSpotResult()

Purpose Update the value of the control and send a message to the Listeners. This is an override of HotSpotResult() in LControl.

SetDescriptor()

Purpose Reset the title and update region of the button. This is an override of SetDescriptor() in LPane.

SetValue()

Purpose Reset button value, update region, and send message. This is an override of SetValue() in LControl.

mText

Purpose This data member contains the text that is drawn in the button.

Access Protected

Prototype
```
LStr255 mText;
```

mTextTraitsID

Purpose This data member contains the resource ID of the resource that contains the text traits information.

Access Protected

Prototype
```
ResIDT mTextTraitsID;
```

mSelectedStyle

Purpose This data member contains the information that determines the style the button text will be drawn with.

Access Protected

Prototype

```
Int16 mSelectedStyle;
```

LTextEdit

Overview

LTextEdit is a PowerPlant class that is used for wrapping the TextEdit functionality of the Mac OS Toolbox.

Methods

The methods in this class are:

LTextEdit()
~LTextEdit()
AdjustCursorSelf()
AdjustImageToText()
AlignTextEditRects()
BeTarget()
ClickSelf()
CreateTextEditStream()
DontBeTarget()
DrawSelf()
FindCommandStatus()
FocusDraw()
GetMacTEH()
GetTextHandle()
HandleKeyPress()
HasAttribute()
HideSelf()
InitTextEdit()
MoveBy()
ObeyCommand()
ResizeFrameBy()
RestorePlace()
SavePlace()
SaveStateForUndo()
ScrollImageBy()
SelectAll()
SetTextHandle()
SetTextPtr()
SetTextTraitsID()
SpendTime()
UserChangedText()

Data Members

The data members in this class are:

mTextEditH
mTextTraitsID
mTextAttributes

Operation

LTextEdit allows you to specify the font, style, size, color, and justification of the text you create.

Source files (Pane Classes)

LTextEdit.h

LTextEdit.cp

Ancestors LAttachable

LCommander

LPane

LPeriodical

LView

LTextEdit()

Purpose The constructor creates objects from the passed-in parameters.

Access Public

Prototype

```
LTextEdit();
LTextEdit( const SPaneInfo &inPaneInfo,
     const SViewInfo &inViewInfo,
     Uint16 inTextAttributes,
     ResIDT inTextTraitsID );
LTextEdit( LStream *inStream );
```

Parameters The parameters for these constructors are:

`const SPaneInfo&`	`inPaneInfo`	The Superpane info.
`const SViewInfo &`	`inViewInfo`	The Superview info.
`Uint16`	`inTextAttributes`	The text attributes.

`ResIDT`	`inTextTraitsID`	The text traits.
`LStream*`	`inStream`	The stream to read from.

~LTextEdit()

Purpose The destructor destroys the object.

Access Virtual, Public

Prototype

```
virtual ~LTextEdit();
```

AdjustCursorSelf()

Purpose TextEdit uses the standard I-Beam cursor. This is an override of AdjustCursorSelf() in LPane.

AdjustImageToText()

Purpose This method resizes the image to match the size of the text.

Access Virtual, Public

Prototype

```
virtual void AdjustImageToText();
```

Parameters None

Return None

AlignTextEditRects()

Purpose Align the view and destination rectangles of the Toolbox TextEdit record with the Frame of a TextEdit.

Access Virtual, Protected

Prototype `virtual void AlignTextEditRects();`

Parameters None

Return None

BeTarget()

Purpose TextEdit is becoming the Target.

Access Virtual, Protected

Prototype `virtual void BeTarget();`

Parameters None

Return None

ClickSelf()

Purpose Respond to Click inside a TextEdit. This is an override of ClickSelf() in LPane.

CreateTextEditStream()

Purpose Create a new TextEdit object from the data in a Stream.

Access Static, Public

Prototype
```
static LTextEdit* CreateTextEditStream(
     LStream *inStream );
```

Parameters The parameters for these constructors are:

`LStream*`	`inStream`	The stream to read from.

Return The pointer to the LTextEdit object that is created.

DontBeTarget()

Purpose TextEdit is no longer the Target. Remove TextEdit from the Idle-Queue. This is an override of DontBeTarget() in LCommander.

DrawSelf()

Purpose Draw a TextEdit. This is an override of DrawSelf() in LPane.

FindCommandStatus()

Purpose This method passes back the status of a command. This is an override of FindCommandStatus() in LCommander.

FocusDraw()

Purpose Prepare for drawing in the TextEdit. This is an override of FocusDraw() in LPane.

GetMacTEH()

Purpose Retrieve theHandle to the Mac TextEdit Record associated with a TextEdit.

Caller may change record fields, and is responsible for redrawing the TextEdit as necessary to reflect any changes. However, caller must not dispose of the TEHandle.

Access Public

Prototype

```
TEHandle GetMacTEH();
```

Parameters None

Return Handle to the Mac TextEdit Record associated with a TextEdit.

GetTextHandle()

Purpose Retrieve a Handle to the text in the LTextEdit. The Handle is the actual Handle used by the Toolbox TextEdit record. Treat this Handle as read-only.

Access Virtual, Public

Prototype
```
virtual Handle GetTextHandle();
```

Parameters None

Return Return a Handle to the text in the LTextEdit.

HandleKeyPress()

Purpose Handle key stroke directed at an TextEdit. This is an override of HandleKeyPress() in LCommander.

HasAttribute()

Purpose Check for the presence of a specific attribute.

Access Public

Prototype
```
Boolean HasAttribute( Uint16 inAttribute );
```

Parameters The parameter for this method is:

`Uint16`	`inAttribute`	The attribute to check for.

Return `Boolean` indicating whether the attribute is present, `false` if not.

HideSelf()

Purpose Hide an LTextEdit. An invisible LTextEdit can't be OnDuty. This is an override of HideSelf() in LPane.

InitTextEdit()

Purpose Initialize member variables of a TextEdit to default values.

Access Private

Prototype

```
void InitTextEdit( ResIDT inTextTraitsID );
```

Parameters The parameter for this method is:

`ResIDT`	`inTextTraitsID`	The text traits to set.

Return None

MoveBy()

Purpose Move the location of the Frame by the specified amounts `inHorizDelta` and `inVertDelta` specify, in pixels, how far to move the Frame (within its surrounding Image). Positive horizontal deltas move to the right, negative to the left. Positive vertical deltas move down, negative up. This is an override of MoveBy() in LPane.

ObeyCommand()

Purpose Issue a command to a Commander. This is an override of ObeyCommand() in LCommander.

ResizeFrameBy()

Purpose Change the Frame size by the specified amounts `inWidthDelta` and `inHeightDelta` specify, in pixels, how much larger to make the Frame. Positive deltas increase the size, negative deltas reduce the size. This is an override of ResizeFrameBy() in LPane.

RestorePlace()

Purpose Restore TextEdit parameters from a stream.

Access Virtual, Public

Prototype
```
virtual void RestorePlace( LStream *inPlace );
```

Parameters The parameter for this method is:

`LStream*`	`inPlace`	The stream to read from.

Return None

SavePlace()

Purpose Save TextEdit parameters to a stream.

Access Virtual, Public

Prototype
```
virtual void SavePlace( LStream *outPlace );
```

Parameters The parameter for this method is:

`LStream*`	`outPlace`	The stream to write to.

Return None

SaveStateForUndo()

Purpose Preserve the state so that an Undo operation is possible.

Access Virtual, Protected

Prototype
```
virtual STextEditUndoH SaveStateForUndo();
```

Parameters None

Return Returns a handle to the Undo information.

ScrollImageBy()

Purpose Scroll the Text. This is an override of ScrollImageBy() in LView.

SelectAll()

Purpose Select entire contents of an TextEdit.

Access Virtual, Public

Prototype
```
virtual void SelectAll();
```

Parameters None

Return None

SetTextHandle()

Purpose Set the text in the LTextEdit to the contents of the specified Handle. The LTextEdit copies the data in the Handle, so the caller retains ownership of the Handle (and should dispose of it as needed).

Access Virtual, Public

Prototype
```
virtual void SetTextHandle( Handle inTextH );
```

Parameters The parameter for this method is:

`Handle`	`inTextH`	The text handle to set to.

Return None

SetTextPtr()

Purpose Set the pointer to the text.

Access Virtual, Public

Prototype

```
virtual void SetTextPtr( Ptr inTextP,
     Int32 inTextLen);
```

Parameters The parameters for this method are:

`Ptr`	`inTextP`	The pointer to the text.
`Int32`	`inTextLen`	The length of the text.

Return None

SetTextTraitsID()

Purpose Specify the resource ID of the TextTraits for an TextEdit. This method updates the line height to fit the text characteristics.

Access Virtual, Public

Prototype

```
virtual void SetTextTraitsID(
     ResIDT inTextTraitsID );
```

Parameters The parameter for this method is:

`ResIDT`	`inTextTraitsID`	The resource ID for the text traits.

Return None

SpendTime()

Purpose Flash the insertion cursor during idle time. This is an override of SpendTime() in LPeriodical.

UserChangedText()

Purpose Text of TextEdit has changed as a result of user action.

You should override this to validate the field and/or dynamically update as the user types. This function is not called by `SetDescriptor()`, which is typically used to programatically change the text.

Access Virtual, Public

Prototype
```
virtual void UserChangedText();
```

Parameters None

Return None

mTextEditH

Purpose Storage for the handle to the TextEdit record.

Access Protected

Prototype
```
TEHandle mTextEditH;
```

mTextTraitsID

Purpose Storage for the text traits.

Access Protected

Prototype
```
ResIDT mTextTraitsID;
```

mTextAttributes

Purpose Storage for the text attributes.

Access Protected

Prototype
```
Uint16 mTextAttributes;
```

LToggleButton

Overview

LToggleButton is a PowerPlant class that implements a button. The graphic for the button needs to be stored as a resource of one of these types:

- ICN#
- ICON'
- 'PICT'

This class is very similar to LButton, except that you may specify five separate resource ID numbers, not just two resource ID numbers as an LButton would have.

Methods

The methods in this class are:

LToggleButton()	DrawGraphic()
DrawSelf()	HotSpotAction()
HotSpotResult()	PointIsInFrame()
SetGraphics()	SetGraphicsType()
SetValue()	

Data Members

The data members in this class are:

mGraphicsType	mOnID
mOnClickID	mOffID
mOffClickID	mTransitionID

Operation

You can use LToggleButton to make some simple animations that play when a button is clicked. For example, you could create a drop-down flag button, or a door that opens and closes when the button is clicked. In addition to the "on" and "off" states of the button that LButton uses, you also specify graphics for:

- A click on a button that is already "on"
- A click on a button that is already "off," and

- A transition graphic that displays when the button is switching states.

Source files (Pane Classes)

`LToggleButton.h`

`LToggleButton.cp`

Ancestors LBroadcaster

LControl

LPane

See also LButton

LToggleButton()

Purpose The constructor creates the objects from the passed-in parameters.

Access Public

Prototype

```
LToggleButton();
LToggleButton( const SPaneInfo &inPaneInfo,
     MessageT inClickedMessage,
     OSType inGraphicsType,
     ResIDT inOnID,
     ResIDT inOnClickID,
     ResIDT inOffID,
     ResIDT inOffClickID,
     ResIDT inTransitionID );
```

Parameters These constructors have the following parameters:

`const LButton&`	`inOriginal`	A reference to the LButton object you want to copy.
`const SPaneInfo&`	`inPaneInfo`	A reference to the SPaneInfo object that is the super view.

`MessageT`	`inClickedMessage`	The message sent when the button is depressed.
`OSType`	`inGraphicsType`	Graphics Type ('ICN#', or 'ICON', or 'PICT')
`ResIDT`	`inOnID`	Resource ID for “on” graphic
`ResIDT`	`inOnClickID`	Resource ID for the “clicked and on” graphic
`ResIDT`	`inOffID`	Resource ID for “off” graphic
`ResIDT`	`inOffClickID`	Resource ID for the “clicked and off” graphic
`ResIDT`	`inTransitionID`	Resource ID for the transition graphic
`LStream*`	`inStream`	A pointer to a stream object that contains the information to create the LButton object.

DrawGraphic()

Purpose Draw the graphic for a the button. The Pane must already be focused before calling this method.

Access Virtual, Protected

Prototype

```
virtual void DrawGraphic( ResIDT inGraphicID );
```

Parameters This method has the following parameter:

`ResIDT`	`inGraphicID`	Resource ID for the button graphic to draw.

Return None

DrawSelf()

Purpose This method is an override of the base class method DrawSelf() in LPane. It draws the button.

HotSpotAction()

Purpose This method is an override of the base class method HotSpotAction() in LControl. It causes the buttons to toggle between two graphics, depending on whether the mouse is inside or outside the button.

HotSpotResult()

Purpose This method is an override of the base class method HotSpotResult() in LControl. It broadcasts a message to Listeners when the button is clicked.

PointIsInFrame()

Purpose This method is an override of the base class method PointIsInFrame() in LPane. It gives you information about whether a point lies within a given frame.

SetGraphics()

Purpose Specify the resources IDs for all states of the button.

Access Virtual, Public

Prototype

```
virtual void SetGraphics( ResIDT inOnID,
     ResIDT inOnClickID,
     ResIDT inOffID,
     ResIDT inOffClickID,
     ResIDT inTransitionID);
```

Parameters This method has the following parameters:

`ResIDT`	`inOnID`	Resource ID for the “on” graphic.
`ResIDT`	`inOnClickID`	Resource ID for the “clicked when on” graphic.

`ResIDT`	`inOffID`	Resource ID for the "off" graphic.
`ResIDT`	`inOffClickID`	Resource ID for the "clicked when off" graphic.
`ResIDT`	`inTransitionID`	Resource ID for the transition graphic.

Return None

SetGraphicsType()

Purpose This method sets the value of the mGraphicsType data member.

Access Virtual, Public

Prototype

```
virtual void SetGraphicsType(
     OSType inGraphicsType );
```

Parameters This method has the following parameter:

`OSType`	`inGraphicsType`	The value to set for mGraphicsType.

SetValue()

Purpose This method is an override of the base class method SetValue() in LControl. It sets the value of the button to on or off.

mGraphicsType

Purpose This method holds the resource type descriptor for the graphic elements of the button. It should be one of the following:

- ICN#
- ICON'
- 'PICT'

Access Protected

Prototype
```
OSType mGraphicsType;
```

mOnID

Purpose This data member holds the resource ID for the "on" button graphic.

Access Protected

Prototype
```
ResIDT mOnID;
```

mOnClickID

Purpose This data member holds the resource ID for the "clicked when on" button graphic.

Access Protected

Prototype
```
ResIDT mOnClickID;
```

mOffID

Purpose This data member holds the resource ID for the "off" button graphic.

Access Protected

Prototype
```
ResIDT mOffID;
```

mOffClickID

Purpose This data member holds the resource ID for the "clicked when off" button graphic.

Access Protected

Prototype `ResIDT mOffClickID;`

mTransitionID

Purpose This data member holds the resource ID for the transition button graphic.

Access Protected

Prototype `ResIDT mTransitionID;`

LUDPEndpoint

Overview LUDPEndpoint is a PowerPlant class that is used for implementing Internet UDP endpoints. UDP is the sessionless protocol of the Internet, and is used by a handful of protocols such as NTP (network time protocol).

Methods The methods in this class are:

LUDPEndpoint()	~LUDPEndpoint()
ReceiveFrom()	SendPacketData()

Data Members There are no data members in this class.

Operation In PowerPlant, there are two subclasses of LUDPEndpoint, named LMacUDPEndpoint and LOpenTptUDPEndpoint. The appropriate class is created automatically when you call UNetworkFactory::CreateUDPEndpoint().

Source files (Networking Classes)

`LUDPEndpoint.h`

`LUDPEndpoint.cp`

See also LEndpoint

LMacTCPUDPEndpoint

LOpenTptUDPEndpoint

LUDPEndpoint()

Purpose The constructor creates the object.

Access Public

Prototype

```
LUDPEndpoint();
```

Parameters None

~LUDPEndpoint()

Purpose The destructor destroys the object.

Access Virtual, Public

Prototype

```
~LUDPEndpoint();
```

ReceiveFrom()

Purpose Receive UDP datagrams.

Access Pure virtual, Public

Prototype

```
void ReceiveFrom( LInternetAddress&
     outRemoteAddress,
     void* outDataBuffer,
     Uint32& ioDataSize,
     Uint32 inTimeoutTicks = Timeout_None) = 0;
```

Parameters The parameter for these constructors is:

`LInternetAddress`	`outRemoteAddress`	The pointer to the thread.
`void*`	`outDataBuffer`	The buffer.
`Uint32&`	`ioDataSize`	The size of the data.
`Uint32`	`inTimeoutTicks`	The time to wait. The default is `Timeout_None`.

Return None

SendPacketData()

Purpose Send UDP datagrams.

Access Pure virtual, Public

Prototype

```
void SendPacketData(
     LInternetAddress& inRemoteHost,
     void* inData,
     Uint32 inDataSize) = 0;
```

Parameters The parameter for these constructors is:

`LInternetAddress`	`inRemoteHost`	The address.
`void*`	`inData`	The buffer.
`Uint32&`	`inDataSize`	The size of the data.

Return None

LUndoer

Overview

LUndoer is a PowerPlant class that helps manage the do, undo, and redo operations. It works in conjunction with LAction.

Methods

The methods in this class are:

LUndoer()
~LUndoer()
ExecuteSelf()
FindUndoStatus()
PostAction()
ToggleAction()
operator=()

Data Members

The data members in this class are:

mAction

Operation

In a typical application having a single-level undo, you should not have to call the LUndoer member functions directly.

Source files

(Action Classes)

`LUndoer.h`

`LUndoer.cp`

Ancestors

LAttachment

See Also

LAction

LAttachment

LCommander

LUndoer()

Purpose

The default constructor initializes the data member. The copy constructor provides a definition so that the compiler does not supply a default copy constructor for this object.

Access Public (default constructor) and Private (copy constructor).

Prototype
```
LUndoer(); /* default constructor */
LUndoer( const LUndoer &inOriginal ); /* pvt copy
     */
```

Parameters None

~LUndoer()

Purpose The destructor destroys the object.

Access Public, Virtual

Prototype
```
virtual ~LUndoer();
```

Parameters None

ExecuteSelf()

Purpose This method intercepts messages for the host. When called with a `msg_PostAction` message it calls SetExecuteHost() with the new action to perform. When called with a `msg_CommandStatus` message it enables and sets the text for the Undo menu item using FindUndoStatus(). When called with a `cmd_Undo` message it toggles betweenUndoing/Redoing the Action using ToggleAction(). If called with any other message, SetExecuteHost() is called with a value of `true`.

Access Virtual, Protected

Prototype
```
virtual void ExecuteSelf( MessageT inMessage,
     void *ioParam);
```

Parameters This method has the following parameters:

`MessageT`	`inMessage`	This is the message that is being received.
`void*`	`ioParam`	This is a pointer to a parameter that accompanies the message.

Return None

FindUndoStatus()

Purpose This method will enable/disable and set the text for the Undo menu item. The menu item can be set to the Redo string, Undo string, or Can't Undo string depending on whether mAction is undoable or redoable.

Access Virtual, Protected

Prototype

```
virtual void FindUndoStatus( SCommandStatus
     *ioStatus);
```

Parameters This method has the following parameter:

`SCommandStatus*`	`ioStatus`	This is a pointer to a structure containing status information for the command.

Return None

PostAction()

Purpose This method handles a new Action that is posted.

Access Virtual, Protected

Prototype

```
virtual void PostAction( LAction *inAction );
```

Parameters This method has the following parameter:

`LAction*`	`inAction`	This is a pointer to LAction object that indicates an Action to be taken.

Return None

ToggleAction()

Purpose Undo/Redo the Action associated with this Undoer, held in the member mAction.

Access Virtual, Protected

Prototype

```
virtual void ToggleAction();
```

Parameters None

Return None

operator=()

Purpose Overloads the assignment operator so that the compiler will not substitute a default.

Access Private

Prototype

```
LUndoer& operator=( const LUndoer &inOriginal );
```

Parameters This operator takes the following parameter:

`const LUndoer&`	`inOriginal`	A reference to an LUndoer object.

Return Returns a reference to an LUndoer object.

mAction

Purpose This data member stores a pointer to the action object that encapsulates the user's most recent action. Because the Undoer ot can send messages to the action object telling it to redo or undo itself.

Access Protected

Prototype

```
LAction *mAction;
```

LVariableArray

Overview

LVariableArray is a PowerPlant class that is similar to LArray, but allows you to store data of differing element sizes.

Methods

The methods in this class are:

LVariableArray()	~LVariableArray()
AdjustAllocation()	AdjustStorage()
AssignItemsAt()	GetItemOffset()
GetItemPtr()	GetItemSize()
GetOffsetsHandle()	GrabItemRangeSize()
GrabItemSize()	InternalAdjustAllocation()
InternalCopyItem()	PokeItem()
ShiftItems()	Sort()
StoreNewItems()	

Data Members

The data members in this class are:

mItemOffsetsH	mItemsAllocated

Operation

This class overrides several member functions of LArray to implement data storage and retrieval in a situation where array elements vary in size.

The public interface and usage is mostly identical to that of LArray.

Source files

(Array Classes)

`LVariableArray.h`

`LVariableArray.cp`

Ancestors

LArray

See Also LArrayIterator

LComparator

LLockedArrayIterator

LVariableArray()

Purpose The constructor creates objects from the passed-in parameters.

Access Public

Prototype

```
LVariableArray( LComparator *inComparator,
     Boolean inKeepSorted );
LVariableArray( Handle inItemsHandle,
     ArrayOffsetH inOffsetsHandle,
     LComparator *inComparator,
     Boolean inIsSorted,
     Boolean inKeepSorted );
```

Parameters The parameters for these constructors are:

`LComparator*`	`inComparator`	The pointer to the comparator object to use.
`Boolean`	`inKeepSorted`	Whether to keep the array sorted or not.
`Handle`	`inItemsHandle`	A handle to the items to put into the array.
`ArrayOffsetH`	`inOffsetsHandle`	An offset. It is assumed that the number of items for the array is the same as the number of offsets in the handle.
`Boolean`	`inIsSorted`	Whether the elements are sorted or not.

~LVariableArray()

Purpose The destructor destroys the object.

Access Public

Prototype

```
virtual ~LVariableArray();
```

AdjustAllocation()

Purpose Adjust the size of the Handle used to store array items. This is an override of AdjustAllocation() in LArray.

AdjustStorage()

Purpose Called internally when the number of bytes used by Items in the array changes. This is an override of AdjustStorage() in LArray.

AssignItemsAt()

Purpose Assign the same value to items starting at the specified index. This is an override of AdjustStorage() in LArray.

GetItemOffset()

Purpose This method returns the offset into the array for a given item.

Access Protected

Prototype

```
Uint32 GetItemOffset( ArrayIndexT inAtIndex);
```

Parameters This method has the following parameter:

`ArrayIndexT`	`inAtIndex`	The index to calculate an offset for.

Return `Uint32` containing the offset.

GetItemPtr()

Purpose Return a pointer to the start of an item's data within the internal storage Handle. This method performs no error checking on the index.

Access Virtual, Public

Prototype

```
virtual void* GetItemPtr(ArrayIndexT inIndex)
     const;
```

Parameters This method has the following parameter:

`ArrayIndexT`	`inIndex`	The index for the item.

Return The pointer to the array item.

Remarks WARNING: Unless you have called Lock(), the returned pointer points to data within a relocatable Handle block.

GetItemSize()

Purpose Return the size in bytes of an item in the array.

Access Virtual, Public

Prototype

```
virtual Uint32 GetItemSize(
     ArrayIndexT inIndex) const;
```

Parameters This method has the following parameter:

`ArrayIndexT`	`inIndex`	The index to get an item.

Return Uint32 containing the size in bytes of an item.

GetOffsetsHandle()

Purpose Return Handle used to store offsets to the data for array items. This is an accessor method for mItemOffsetsH.

Access Public

Prototype
```
Handle GetOffsetsHandle() const;
```

Parameters None

Return The handle used to store offsets (mItemOffsetsH).

Remarks
- Treat the Handle as read-only. Changing the data in the Handle could invalidate the internal state of the array.
- In general, you should only access the offsets Handle in order to copy it (either in memory or by writing it to a file).

GrabItemRangeSize()

Purpose Return the size in bytes of the items from `inStartIndex` to `inEndIndex`, inclusive. This is an override of GrabItemRangeSize() in LArray.

GrabItemSize()

Purpose Return the size in bytes of the specified item. This method is the same as GetItemSize() except it doesn't validate `inIndex`. This is an override of GrabItemSize() in LArray.

InternalAdjustAllocation()

Purpose Called internally to change the size of the storage used. This is an override of InternalAdjustAllocation() in LArray.

InternalCopyItem()

Purpose Set the item at `inDestIndex` to a copy of the item at `inSourceIndex`. This is an override of InternalCopyItem() in LArray.

PokeItem()

Purpose Store data for the item at the specified index. This is an override of PokeItem() in LArray.

ShiftItems()

Purpose Moves items within the Handle used for internal storage. This is an override of ShiftItems() in LArray.

Sort()

Purpose Sort items in the array. This is an override of Sort() in LArray.

StoreNewItems()

Purpose Store values within the internal storage Handle. Items all have the same value, and space must already have been allocated for them. This is an override of StoreNewItems() in LArray.

mItemOffsetsH

Purpose This stores the handle for the offsets.

Access Protected

Prototype
```
ArrayOffsetH mItemOffsetsH;
```

mItemsAllocated

Purpose This stores the items allocated information.

Access Protected

Prototype

```
Uint32 mItemsAllocated;
```

LView

Overview

LView is a PowerPlant class that is the basis for the visual hierarchy. An LView object is a pane that can contain other panes.

Methods

The methods in this class are:

LView()
~LView()
Activate()
AdaptToNewSurroundings()
AdaptToSuperFrameSize()
AddSubPane()
AdjustCursor()
AutoScrollImage()
CalcPortOrigin()
CalcRevealedRect()
Click()
CountPanels()
Deactivate()
DeleteAllSubPanes()
Disable()
DontRefresh()
Draw()
Enable()
EstablishPort()
ExpandSubPane()
FindConstPaneByID()
FindDeepSubPaneContaining()
FindPaneByID()
FindShallowSubPaneContaining()
FinishCreate()
FocusDraw()
FocusDraw()
FocusExposed()
GetDescriptorForPaneID()
GetImageLocation()
GetDescriptorForPaneID()
GetImageLocation()
GetImageSize()
GetInFocusView()
GetLocalUpdateRgn()
GetPortOrigin()
GetRevealedRect()
GetScrollPosition()
GetScrollUnit()
GetSubPanes()
GetValueForPaneID()
Hide()

ImagePointIsInFrame()
ImageRectIntersectsFrame()
ImageToLocalPoint()
InitView()
LocalToImagePoint()
LocalToPortPoint()
MoveBy()
OrientAllSubPanes()
OrientSubPane()
OutOfFocus()
PortToLocalPoint()
PrintPanel()
ReconcileFrameAndImage()
Refresh()
RemoveSubPane()
ResizeFrameBy()
ResizeImageBy()
ResizeImageTo()
RestorePlace()
SavePlace()
ScrollBits()
ScrollImageBy()
ScrollImageTo()
ScrollPinnedImageBy()
ScrollToPanel()
SetDescriptorForPaneID()
SetReconcileOverhang()
SetScrollUnit()
SetValueForPaneID()
Show()
SubImageChanged()
SuperActivate()
SuperDeactivate()
SuperDisable()
SuperEnable()
SuperHide()
SuperPrintPanel()
SuperShow()

Data Members

The data members in this class are:

sInFocusView
mImageSize
mImageLocation
mScrollUnit
mPortOrigin
mSubPanes

mRevealedRect mUpdateRgnH

mReconcileOverhang

Operation

LView objects contain the panes that you draw. In other words, LView is a container object, with the panes being the contents. Objects that derive from LView are usually places where you draw things.

To learn more about views, refer to *The PowerPlant Book*.

Source files

(Pane Classes)

`LView.h`

`LView.cp`

Ancestors

LAttachable

LPane

LView()

Purpose

The constructors create objects from the passed-in parameters.

Access

Public

Prototype

```
LView();
LView( const LView &inOriginal );
LView( const SPaneInfo &inPaneInfo,
     const SViewInfo &inViewInfo );
LView( LStream *inStream );
```

Parameters

The parameters for the constructors are:

`const LView&`	`inOriginal`	The LView object to copy.
`const SPaneInfo&`	`inPaneInfo`	The reference to the Superpane.

`const SViewInfo&`	`inViewInfo`	The reference to the Superview.
`LStream*`	`inStream`	The stream to construct with.

~LView()

Purpose The destructor destroys the object.

Access Virtual, Public

Prototype
```
virtual ~LView();
```

Activate()

Purpose Activate a View. This is an override of Activate() in LPane.

AdaptToNewSurroundings()

Purpose Adjust view when its SuperView changes identity or size. This is an override of AdaptToNewSurroundings() in LPane.

AdaptToSuperFrameSize()

Purpose Adjust state of View when size of SuperView's Frame changes by the specified amounts. This is an override of AdaptToSuperFrameSize() in LPane.

AddSubPane()

Purpose You should not normally call this function. Call LPane::PutInside() to associate a Pane with a SuperView.

Access Virtual, Public

Prototype
```
virtual void AddSubPane( LPane *inSub );
```

Parameters The parameter for this method is:

`LPane*`	`inSub`	The pointer to the sub pane to add.

Return None

AdjustCursor()

Purpose Wrapper function for setting the cursor shape. View needs to determine which subpane, if any, should set the cursor shape. This is an override of AdjustCursor() in LPane.

AutoScrollImage()

Purpose Scroll the image if the specified point is outside the View's Frame.

Call this function while tracking the mouse to scroll a View in the direction of the mouse location.

Access Virtual, Public

Prototype
```
virtual Boolean AutoScrollImage( Point inLocalPt );
```

Parameters The parameter for this method is:

`Point`	`inLocalPt`	The pointer to the sub pane to add.

Return Returns `true` if the View actually scrolled, else `false`.

CalcPortOrigin()

Purpose Calculate the coordinate origin for the Port needed to set up the local coordinates of a View.

Access Virtual, Public

Prototype
```
virtual void CalcPortOrigin();
```

Parameters None

Return None

Remarks The Port origin must be in 16-bit space. In fact, the limitation is more restrictive because the entire Port must be in 16-bit space. The origin is the top left corner. To make sure that the bottom right corner is in 16-bit space, we force the origin to be less than 2^14 (16,384), which allows Port dimensions of a maximum of 16,384 pixels. At 72 dpi, this is about 227 inches or 19 feet (much more screen or printer area than you can get with current technology).

This means that Local and Image coordinates will be the same when the Image size is less than 16,384 pixels. For Images greater than this, you can't use absolute coordinates for drawing. You need to offset the coordinates using ImageToLocalPoint() and LocalToImagePoint().

The true coordinates offset is the distance between the top left corners of the Image and the Port. If this offset is greater than 2^14, we use an effective offset that is the true offset modulo 2^14:

effective_offset = true_offset modulo 2^14

Using this effective offset maintains the bit-wise alignment of the Port with respect to base-2 byte boundaries. This is important for drawing Toolbox Patterns and PixPats, as well as for `CopyBits()` calls.

CalcRevealedRect()

Purpose Calculate the portion of the Frame which is revealed through the frames of all SuperViews. mRevealedRect is in Port coordinates.

Access Public

Prototype

```
void CalcRevealedRect();
```

Parameters None

Return None

Click()

Purpose Handle a click inside a View. This is an override of Click() in LPane.

CountPanels()

Purpose Return the number of horizontal and vertical Panels. A Panel is a "frameful" of a View's Image. This is an override of CountPanels() in LPane.

Deactivate()

Purpose Deactivate a View. This is an override of Deactivate() in LPane.

DeleteAllSubPanes()

Purpose Deletes all subpanes of a View.

Access Public

Prototype
```
void DeleteAllSubPanes();
```

Parameters None

Return None

Disable()

Purpose Disable a View. This is an override of Disable() in LPane.

DontRefresh()

Purpose Validate the area occupied by a View. This removes the View area from the update region so that the View won't be redrawn during the next Update event.

This is an override of DontRefresh() in LPane.

Draw()

Purpose Draw a View and all its subpanes. This is an override of Draw() in LPane.

Enable()

Purpose Enable a View. This is an override of Enable() in LPane.

EstablishPort()

Purpose Set the GrafPort for a View.

Access Virtual, Public

Prototype
```
virtual Boolean EstablishPort();
```

Parameters None

Return Return `true` if the port was set successfully, else `false`.

ExpandSubPane()

Purpose Adjust the size and location of a SubPane to fit within the interior of a View in the horizontal and/or vertical directions.

Access Virtual, Public

Prototype
```
virtual void ExpandSubPane( LPane *inSub,
     Boolean inExpandHoriz,
     Boolean inExpandVert);
```

Parameters The parameters for this method are:

`LPane*`	`inSub`	The pointer to the sub pane to add.
`Boolean`	`inExpandHoriz`	Tell whether to expand horizontally.
`Boolean`	`inExpandVert`	Tell whether to expand vertically.

Return None

FindConstPaneByID()

Purpose Find the Pane of a View which has the specified ID. This is an override of FindConstPaneByID() in LPane.

FindDeepSubPaneContaining()

Purpose Return the most deeply nested visible SubPane which contains the specified point, which is in Port coordinates. This is an override of FindDeepSubPaneContaining() in LPane.

FindPaneByID()

Purpose Find the Pane of a View which has the specified ID. This is an override of FindPaneByID() in LPane.

FindShallowSubPaneContaining()

Purpose Return the immediate visible SubPane which contains the specified point, which is in Port coordinates. This is an override of FindShallowSubPaneContaining() in LPane.

FindSubPaneHitBy()

Purpose Find the SubPane of this View that is hit by the specified point. This is an override of FindSubPaneHitBy() in LPane.

FinishCreate()

Purpose Wrapper function for FinishCreateSelf. This is an override of FinishCreate() in LPane.

You will rarely want to override this function.

FocusDraw()

Purpose Prepare for drawing in the View by setting the Port and clipping area. This is an override of FocusDraw() in LPane.

FocusExposed()

Purpose Prepare for drawing in the View by setting the Port and clipping area. This is an override of FocusExposed() in LPane.

GetDescriptorForPaneID()

Purpose Get the Descriptor of the Pane of a View which has the specified ID. The Pane may be the View itself, or a subpane of the View.

Access Virtual, Public

Prototype

```
virtual StringPtr GetDescriptorForPaneID(
     PaneIDT inPaneID,
     Str255 outDescriptor) const;
```

Parameters The parameters for this method are:

`PaneIDT`	`inPaneID`	The resource ID for the pane.
`Str255`	`outDescriptor`	The string that we want, the descriptor.

Return Returns a pointer that is the same as `outDescriptor`.

GetImageLocation()

Purpose Determine the location of the image within a View.

Access Public

Prototype
```
void GetImageLocation(SPoint32 &outLocation) const;
```

Parameters The parameter for this method is:

`SPoint32&`	`outLocation`	The location of the image within the view.

Return None

GetImageSize()

Purpose Pass back the dimensions of a View's image.

Access Virtual, Public

Prototype
```
virtual void GetImageSize(
     SDimension32 &outSize) const;
```

Parameters The parameter for this method is:

`SDimension32&`	`outSize`	The dimensions of the image within the view.

Return None

GetInFocusView()

Purpose Retrieve a pointer to the view that has the focus. This is an accessor method for the sInFocusView data member.

Access Static, Public

Prototype
```
static LView* GetInFocusView();
```

Parameters None

Return The pointer to the LView that has the focus.

GetLocalUpdateRgn()

Purpose Return the region, in local coordinates, to which update drawing is clipped. This is an override of GetLocalUpdateRgn() in LPane.

GetPortOrigin()

Purpose Pass back the coordinate origin for the Port needed to set up the local coordinates of a View.

Access Virtual, Public

Prototype
```
virtual void GetPortOrigin(Point &outOrigin) const;
```

Parameters The parameter for this method is:

`Point&`	`outOrigin`	The coordinate origin passed back to the caller.

Return None

GetRevealedRect()

Purpose Retrieve the Rect from mRevealedRect.

Access Public

Prototype

```
void GetRevealedRect( Rect &outRect ) const;
```

Parameters The parameter for this method is:

`Rect&`	`outRect`	The Rect in mRevealedRect is passed back to the caller.

Return None

GetScrollPosition()

Purpose Pass back the location of a View's frame within its image.

Access Virtual, Public

Prototype

```
virtual void GetScrollPosition(
     SPoint32 &outScrollPosition) const;
```

Parameters The parameter for this method is:

`SPoint32&`	`outScrollPosition`	The location of the View's frame.

Return None

GetScrollUnit()

Purpose This method retrieves the value of mScrollUnit.

Access Public

Prototype

```
void GetScrollUnit(
     SPoint32 &outScrollUnit ) const;
```

Parameters The parameter for this method is:

`SPoint32&`	`outScrollUnit`	The value of mScrollUnit.

Return None

GetSubPanes()

Purpose This method returns the value of mSubPanes.

Access Public

Prototype

```
TArray<LPane*>& GetSubPanes();
```

Parameters None

Return A reference to an array that contains mSubPanes.

GetValueForPaneID()

Purpose Get the Value of the Pane of a View which has the specified ID. The Pane may be the View itself, or a subpane of the View.

Access Virtual, Public

Prototype

```
virtual Int32 GetValueForPaneID(
     PaneIDT inPaneID ) const;
```

Parameters The parameter for this method is:

`PaneIDT`	`inPaneID`	The pane ID to query the value of.

Return Returns the value in an `Int32`.

Hide()

Purpose Make a View invisible. This is an override of Hide() in LPane.

ImagePointIsInFrame()

Purpose Return whether a Point specified in Image Coords is within the Frame of a View.

Access Public

Prototype

```
Boolean ImagePointIsInFrame(
     Int32 inHorizImage,
     Int32 inVertImage ) const;
```

Parameters The parameters for this method are:

`Int32`	`inHorizImage`	The horizontal coordinate.
`Int32`	`inVertImage`	The vertical coordinate.

Return `Boolean` indicating whether the point is in the frame (`true`) or not (`false`).

ImageRectIntersectsFrame()

Purpose Determine whether a rectangle specified in image coordinates intersects the frame of a View.

Access Public

Prototype

```
Boolean ImageRectIntersectsFrame(
     Int32 inLeftImage,
     Int32 inTopImage,
     Int32 inRightImage,
     Int32 inBottomImage ) const;
```

Parameters The parameters for this method are:

`Int32`	`inLeftImage`	The left coordinate.
`Int32`	`inTopImage`	The top coordinate.
`Int32`	`inRightImage`	The right coordinate.
`Int32`	`inBottomImage`	The bottom coordinate.

Return `Boolean` indicating whether the rectangle is in the frame (`true`) or not (`false`).

ImageToLocalPoint()

Purpose Convert point from Image (32-bit) to Local (16-bit) coordinates.

Image and Local coordinates are different only when the Image size is greater than 16K (15-bit).

Access Public

Prototype

```
void ImageToLocalPoint(
     const SPoint32 &inImagePt,
     Point &outLocalPt ) const;
```

Parameters The parameters for this method are:

`const SPoint32&`	`inImagePt`	The point to convert.
`Point&`	`outLocalPt`	The converted point.

Return None

InitView()

Purpose Initializer for the class.

Access Protected

Prototype

```
void InitView( const SViewInfo &inViewInfo );
```

Parameters The parameter for this method is:

`const SViewInfo&`	`inViewInfo`	The reference to the Superview to initialize.

Return None

LocalToImagePoint()

Purpose Convert point from Local (16-bit) to Image (32-bit) coordinates.

Image and Local coordinates are different only when the Image size is greater than 16K (15-bit)

Access Public

Prototype

```
void LocalToImagePoint(
     const Point &inLocalPt,
     SPoint32 &outImagePt ) const;
```

Parameters The parameters for this method are:

`const Point&`	`inLocalPt`	The point to convert.
`SPoint32&`	`outImagePt`	The converted point.

Return None

LocalToPortPoint()

Purpose Convert point from Local to Port coordinates. This is an override of LocalToPortPoint() in LPane.

MoveBy()

Purpose Move the location of the Image and Frame by the specified amounts. This is an override of MoveBy() in LPane.

OrientAllSubPanes()

Purpose Adjust the Enabled, Active, and Visible properties of all subpanes.

This function is for the convenience of people who have code from CW5 (PowerPlant 1.0.2) and earlier which created Panes directly in the code (not from 'PPob' resources). You really should call FinishCreate() for each Pane, but to quickly get code running, you can call this function after creating several Panes inside one View.

Do NOT use this function for new code.

Access Public

Prototype

```
void OrientAllSubPanes();
```

Parameters None

Return None

OrientSubPane()

Purpose Adjust the Enabled, Active, and Visible properties of a subpane based on the properties of this View (which must be the subpane's SuperView).

Access Virtual, Public

Prototype

```
virtual void OrientSubPane( LPane *inSub );
```

Parameters The parameter for this method is:

`LPane*`	`inSub`	The pointer to the subpane.

Return None

OutOfFocus()

Purpose Notify the View system that a View is no longer in focus.

Call when the coordinate system or clipping region of a View changes, passing a pointer to that View. This clears the saved focus if that View was the one in focus.

Use `nil` for inView if you manually change (and don't restore) the current port or clipping region

Access Static, Public

Prototype

```
static void OutOfFocus( LView *inView );
```

Parameters The parameter for this method is:

`LView*`	`inView`	The pointer to the view that is no longer in focus.

Return None

PortToLocalPoint()

Purpose Convert a point from Port to local coordinates. This is an override of PortToLocalPoint() in LPane.

PrintPanel()

Purpose Try to print a panel of a View. The View is at the top level of the Printout, meaning that it controls pagination. This functions scrolls the View to the specified panel.

This is an override of PrintPanel() in LPane.

ReconcileFrameAndImage()

Purpose

Adjusts the Image so that it fits within the Frame. This function addresses the problem of what to do when you scroll a View to at or near the bottom or right, then make the View's frame larger. This would normally expose some undefined area below or to the right of the image.

If mReconcileOverhang is true, this function scrolls the image so that the bottom right corner is at the bottom right of the frame. However, it never moves the top left corner of the image beyond the top left of the frame. Therefore, the only time undefined area is exposed is when the frame is larger than the Image.

For Views with fixed image sizes, such as drawings where the Image size is the size of a printed page, set mReconcileOverhang to `true`. The user does not normally want to see past the bottom or right of such Views.

For Views with variable Image sizes, such as text blocks where the size of the Image depends on the number of lines of text, set mReconcileOverhang to `false`. The user may want to see the undefined area in anticipation of the image growing.

Access

Virtual, Protected

Prototype

```
virtual void ReconcileFrameAndImage(
     Boolean inRefresh );
```

Parameters

The parameter for this method is:

`Boolean`	`inRefresh`	The value indicating whether to refresh.

Return

None

Refresh()

Purpose Invalidate the area occupied by a View. This forces an Update event that, when processed, will redraw the View. Refresh does nothing if the View is not visible. This is an override of Refresh() in LPane.

RemoveSubPane()

Purpose You should not normally call this function. Call LPane::PutInside(nil) to remove a SubPane from its SuperView

Access Virtual, Public

Prototype

```
virtual void RemoveSubPane(
     LPane *inSub);
```

Parameters The parameter for this method is:

`LPane*`	`inSub`	The value indicating whether to refresh.

Return None

ResizeFrameBy()

Purpose Change the Frame size by the specified amounts. This is an override of ResizeFrameBy() in LPane.

ResizeImageBy()

Purpose Change the image size by the specified pixel increments.

Access Public

Prototype

```
virtual void ResizeImageBy(
     Int32 inWidthDelta,
     Int32 inHeightDelta,
     Boolean inRefresh );
```

Parameters The parameters for this method are:

`Int32`	`inWidthDelta`	The value indicating the change in width.
`Int32`	`inHeightDelta`	The value indicating the change in height.
`Boolean`	`inRefresh`	Whether to refresh or not.

Return None

ResizeImageTo()

Purpose Set the image size to the specified pixel dimensions.

Access Public

Prototype

```
void ResizeImageTo(
     Int32 inWidth,
     Int32 inHeight,
     Boolean inRefresh );
```

Parameters The parameters for this method are:

`Int32`	`inWidth`	The value indicating the change in width.
`Int32`	`inHeight`	The value indicating the change in height.
`Boolean`	`inRefresh`	Whether to refresh or not.

Return None

RestorePlace()

Purpose Read size and location information stored in a Stream by the SavePlace() function. This is an override of RestorePlace() in LPane.

SavePlace()

Purpose Write size and location information to a Stream for later retrieval by the RestorePlace() function. This is an override of SavePlace() in LPane.

ScrollBits()

Purpose Scroll the pixels of a View. This method is called internally by ScrollImageBy() to shift the pixels.

Access Virtual, Public

Prototype

```
virtual void ScrollBits(
     Int32 inLeftDelta,
     Int32 inTopDelta ) ;
```

Parameters The parameters for this method are:

`Int32`	`inLeftDelta`	The value indicating the change from the left.
`Int32`	`inTopDelta`	The value indicating the change from the top.

Return None

ScrollImageBy()

Purpose Scroll image by specified horizontal and vertical increments. Scrolling moves the image relative to the frame and port, so that a different portion of the image is visible thru the frame.

Positive deltas scroll right and down. Negative deltas scroll left and up.

If inRefresh is true, the Port containing the View is updated immediately, rather than refreshed at the next update event. Scrolling

usually happens during mouse down tracking, so we want immediate visual feedback.

Access Public

Prototype

```
virtual void ScrollImageBy(
     Int32 inLeftDelta,
     Int32 inTopDelta,
     Boolean inRefresh );
```

Parameters The parameters for this method are:

`Int32`	`inLeftDelta`	The value indicating the change from the left.
`Int32`	`inTopDelta`	The value indicating the change from the top.
`Boolean`	`inRefresh`	Whether to refresh or not.

Return None

ScrollImageTo()

Purpose Scroll image to the specified horizontal and vertical locations.

Scrolling moves the image relative to the frame and port, so that a different portion of the image is visible thru the frame.

When scrolled to (0, 0), the top left of the image coincides with the top left of the frame (home position).

If `inRefresh` is true, the Port containing the View is updated immediately, rather than refreshed at the next update event. Scrolling usually happens during mouse down tracking, so we want immediate visual feedback.

Access Public

Prototype

```
void ScrollImageTo(
     Int32 inLeftLocation,
     Int32 inTopLocation,
     Boolean inRefresh );
```

Parameters The parameters for this method are:

`Int32`	`inLeftLocation`	The value indicating the point on the left.
`Int32`	`inTopLocation`	The value indicating the on the top.
`Boolean`	`inRefresh`	Whether to refresh or not.

Return None

ScrollPinnedImageBy()

Purpose Scroll image by specified horizontal and vertical increments, but don't scroll beyond an edge of the frame.

Access Public

Prototype

```
virtual Boolean ScrollPinnedImageBy(
     Int32 inLeftDelta,
     Int32 inTopDelta,
     Boolean inRefresh );
```

Parameters The parameters for this method are:

`Int32`	`inLeftDelta`	The value indicating the change from the left.
`Int32`	`inTopDelta`	The value indicating the change from the top.
`Boolean`	`inRefresh`	Whether to refresh or not.

Return Return `true` if the View actually scrolls, else `false`.

ScrollToPanel()

Purpose Scroll View Image to the specified panel. This is an override of ScrollToPanel() in LPane.

SetDescriptorForPaneID()

Purpose Set the descriptor of the Pane of a View which has the specified ID. The Pane may be the View itself, or a subpane of the View.

Access Virtual, Public

Prototype

```
virtual void SetDescriptorForPaneID(
     PaneIDT inPaneID,
     ConstStr255Param inDescriptor );
```

Parameters The parameters for this method are:

`PaneIDT`	`inPaneID`	The pane ID.
`ConstStr-255Param`	`inDescriptor`	The string to set for the descriptor.

Return None

SetReconcileOverhang()

Purpose Specify whether to reconcile the Frame and Image when there is overhang. Refer to ReconcileFrameAndImage() to learn more.

Access Public

Prototype

```
void SetReconcileOverhang( Boolean inSetting );
```

Parameters The parameter for this method is:

`Boolean`	`inSetting`	The value to set for mReconcileOverhang.

Return None

SetScrollUnit()

Purpose Set the mScrollUnit data member.

Access Public

Prototype

```
void SetScrollUnit( const SPoint32 &inScrollUnit );
```

Parameters The parameter for this method is:

`const SPoint32&`	`inScrollUnit`	The value to set for mScrollUnit.

Return None

SetValueForPaneID()

Purpose Set the value of the pane of a View which has the specified ID. The pane may be the View itself, or a subpane of the View.

Access Virtual, Public

Prototype

```
virtual void SetValueForPaneID(
     PaneIDT inPaneID,
     Int32 inValue );
```

Parameters The parameters for this method are:

`PaneIDT`	`inPaneID`	The pane ID.
`Int32`	`inValue`	value to set for the pane of the View.

Return None

Show()

Purpose Make a View visible. This is an override of Show() in LPane.

SubImageChanged()

Purpose This method gets called as notification that the image of some SubView changed size, location, or scroll units. You should override this method to respond to such changes.

Access Virtual, Public

Prototype

```
virtual void SubImageChanged( LView* inSubView );
```

Parameters The parameter for this method is:

`LView*`	`inSubView`	The pointer to the SubView.

Return None

SuperActivate()

Purpose The SuperView of a View has been activated. This is an override of SuperActivate() in LPane.

SuperDeactivate()

Purpose The SuperView of a View has been deactivated. This is an override of SuperDeactivate() in LPane.

SuperDisable()

Purpose The SuperView of a View has been disabled. This is an override of SuperDisable() in LPane.

SuperEnable()

Purpose The SuperView of a View has been enabled. This is an override of SuperEnable() in LPane.

SuperHide()

Purpose This is called when the SuperView of a View has been hidden. This is an override of SuperHide() in LPane.

SuperPrintPanel()

Purpose This is called when the SuperView is printing a panel. This is an override of SuperPrintPanel() in LPane.

SuperShow()

Purpose The SuperView of a View has become visible. This is an override of SuperShow() in LPane.

sInFocusView

Purpose The pointer to the LView that has the focus.

Access Protected

Prototype
```
static LView *sInFocusView;
```

mImageSize

Purpose The image size.

Access Protected

Prototype
```
SDimension32 mImageSize;
```

mImageLocation

Purpose The image location.

Access Protected

Prototype
```
SPoint32 mImageLocation;
```

mScrollUnit

Purpose The scroll unit.

Access Protected

Prototype
```
SPoint32 mScrollUnit;
```

mPortOrigin

Purpose The origin for the port.

Access Protected

Prototype
```
Point mPortOrigin;
```

mSubPanes

Purpose A list of the subpanes for this View.

Access Protected

Prototype
```
TArray<LPane*> mSubPanes;
```

mRevealedRect

Purpose The revealed rectangle.

Access Protected

Prototype `Rect mRevealedRect;`

mUpdateRgnH

Purpose The handle to the update region.

Access Protected

Prototype `RgnHandle mUpdateRgnH;`

mReconcileOverhang

Purpose The value indicating whether to reconcile overhang.

Access Protected

Prototype `Boolean mReconcileOverhang;`

LWindow

Overview LWindow is a PowerPlant class that defines window behavior.

Methods The methods in this class are:

LWindow()
~LWindow()
Activate()
ActivateSelf()
ApplyForeAndBackColors()
AttemptClose()
CalcStandardBounds()
CalcStandardBoundsForScreen()
ClearAttribute()
ClickInContent()
ClickInDrag()
ClickInGoAway()
ClickInGrow()
ClickInZoom()
CreateWindow()
Deactivate()
DeactivateSelf()
DoClose()
DoSetBounds()
DoSetPosition()
DoSetZoom()
DrawSelf()
DrawSizeBox()
Enable()
EstablishPort()
ExpandSubPane()
FetchWindowObject()
FindCommandStatus()
FindWindowByID()
GetAEProperty()
GetAEWindowAttribute()
GetDescriptor()
GetGlobalBounds()
GetMacPort()
GetMinMaxSize()
GetStandardSize()
GlobalToPortPoint()
HandleAppleEvent()
HandleClick()
HasAttribute()
HideSelf()
InitWindow()
InvalPortRect()
InvalPortRgn()

MakeMacWindow() MakeSelfSpecifier()
MoveWindowBy() MoveWindowTo()
ObeyCommand() PortToGlobalPoint()
ResizeFrameBy() ResizeWindowBy()
ResizeWindowTo() Resume()
Select() SendAESetBounds()
SendAESetPosition() SendAESetZoom()
SetAEProperty() SetAttribute()
SetDescriptor() SetForeAndBackColors()
SetMinMaxSize() SetStandardSize()
Show() ShowSelf()
Suspend() UpdatePort()
ValidPortRect() ValidPortRgn()

Data Members

The data members in this class are:

mMacWindowP mMinMaxSize
mStandardSize mUserBounds
mAttributes mForeColor
mBackColor mMoveOnlyUserZoom

Operation

LWindow is a complex class that derives from many other classes. Many of the member functions in LWindow are for internal use only, and you shouldn't have to call them directly.

For more information on working with the LWindow class, refer to *The PowerPlant Book*.

Source files

(Pane Classes)

`LWindow.h`

`LWindow.cp`

Ancestors

LAttachable

LCommander

LModelObject

LPane

LView

LWindow()

Purpose The constructor creates objects from the passed-in parameters.

Access Public

Prototype

```
LWindow();
LWindow( const SWindowInfo &inWindowInfo);
LWindow( ResIDT inWINDid,
     Uint32 inAttributes,
     LCommander *inSuper );
LWindow( LStream *inStream );
```

Parameters These constructors have the following parameters:

`const SWindowInfo&`	`inWindowInfo`	The SuperView reference.
`ResIDT`	`inWINDid`	The resource ID for the WIND resource.
`Uint32`	`inAttributes`	The attributes for the window (see `Window.h`).
`LCommander*`	`inSuper`	The pointer to the Supercommander.
`LStream*`	`inStream`	The stream to use to create the object.

~LWindow()

Purpose The destructor destroys the object.

Access Public

Prototype

```
virtual ~LWindow();
```

Activate()

Purpose Activate a Window. This is an override of Activate() in LPane.

ActivateSelf()

Purpose This is called when a Window is being activated. This is an override of ActivateSelf() in LPane.

ApplyForeAndBackColors()

Purpose Set the foreground and background colors of the current port. The Window or one of its subpanes must already be focused. This is an override of ApplyForeAndBackColors() in LPane.

AttemptClose()

Purpose Try to close a Window as a result of direct user action.

Access Virtual, Public

Prototype

```
virtual void AttemptClose();
```

Parameters None

Return None

CalcStandardBounds()

Purpose Calculate the bounds of the Window in its standard state and return whether it is in the standard state. The standard state depends on the screen containing the largest area of the Window and the current standard size.

Access Virtual, Public

Prototype

```
virtual Boolean CalcStandardBounds(
     Rect &outStdBounds) const;
```

Parameters The parameter for this method is:

`Rect&`	`outStdBounds`	The port rectangle of Window at standard size in global coordinates.

Return Returns `true` if in the standard state, else `false`.

CalcStandardBoundsForScreen()

Purpose Calculate the bounds of the Window if it was at a standard (zoomed out) state on a Screen with the specified bounds.

Access Virtual, Public

Prototype

```
virtual void CalcStandardBoundsForScreen(
     const Rect &inScreenBounds,
     Rect &outStdBounds) const;
```

Parameters The parameters for this method are:

`Rect&`	`outStdBounds`	The port rectangle of Window at standard size in global coordinates.
`const Rect&`	`inScreenBounds`	Bounding box of screen in global coordinates.

Return None

ClearAttribute()

Purpose Clear the attribute in mAttributes that is passed in to this method.

Access Public

Prototype

```
void ClearAttribute( EWindAttr inAttribute );
```

Parameters The parameter for this method is:

`EWindAttr`	`inAttribute`	The window attributes.

Return None

ClickInContent()

Purpose Respond to a click in the content region of a Window.

Access Virtual, Public

Prototype

```
virtual void ClickInContent(
     const EventRecord &inMacEvent);
```

Parameters The parameter for this method is:

`const EventRecord&`	`inMacEvent`	The event record for the click.

Return None

ClickInDrag()

Purpose Process a click in drag region of a Window.

Access Virtual, Public

Prototype

```
virtual void ClickInDrag(
     const EventRecord &inMacEvent);
```

Parameters The parameter for this method is:

`const EventRecord&`	`inMacEvent`	The event record for the click.

Return None

ClickInGoAway()

Purpose Handle a click inside the close box of a Window.

Access Virtual, Public

Prototype

```
virtual void ClickInGoAway(
     const EventRecord &inMacEvent)
```

Parameters The parameter for this method is:

`const EventRecord&`	`inMacEvent`	The event record for the click.

Return None

ClickInGrow()

Purpose Handle a click in the Grow box of a Window.

Access Virtual, Public

Prototype

```
virtual void ClickInGrow(
     const EventRecord &inMacEvent);
```

Parameters The parameter for this method is:

`const EventRecord&`	`inMacEvent`	The event record for the click.

Return None

ClickInZoom()

Purpose Handle a click inside the zoom box of a Window

Access Virtual, Public

Prototype

```
virtual void ClickInZoom(
     const EventRecord &inMacEvent,
     Int16 inZoomDirection)
```

Parameters The parameters for this method are:

`const EventRecord&`	`inMacEvent`	The event record for the click.
`Int16`	`inZoomDirection`	The direction to zoom.

Return None

CreateWindow()

Purpose Return a newly created Window object initialized from a PPob resource.

Access Static, Public

Prototype

```
static Window* CreateWindow(
     ResIDT inWindowID,
     LCommander *inSuperCommander);
```

Parameters The parameters for this method are:

`ResIDT`	`inWindowID`	The resource ID for the window.
`LCommander*`	`inSuperCommander`	The pointer to the Supercommander.

Return The pointer to the Window.

Deactivate()

Purpose Deactivate the Window. This is an override of Deactivate() in LPane.

DeactivateSelf()

Purpose This is called when a Window is being deactivated. This is an override of DeactivateSelf() in LPane.

DoClose()

Purpose Close a Window.

Access Virtual, Public

Prototype

```
virtual void DoClose();
```

Parameters None

Return None

DoSetBounds()

Purpose Change size and location of a Window.

Access Virtual, Public

Prototype

```
virtual void DoSetBounds(
     const Rect &inBounds );
```

Parameters The parameter for this method is:

`const Rect&`	`inBounds`	The Rect for the bounds. In global coordinates, this specifies the new size and location of the Window's port rectangle

Return None

DoSetPosition()

Purpose Change the location of a Window.

Access Virtual, Public

Prototype

```
virtual void DoSetPosition( Point inPosition );
```

Parameters The parameter for this method is:

`Point`	`inPosition`	The top left corner of the Window's port rectangle is placed at `inPosition`, which is in global coordinates.

Return None

DoSetZoom()

Purpose Zoom window to either the Standard or User state.

Access Virtual, Public

Prototype

```
virtual void DoSetZoom(
     Boolean inZoomToStdState );
```

Parameters The parameter for this method is:

`Boolean`	`inZoomToStdState`	Indicate whether to zoom to the standard state.

Return None

DrawSelf()

Purpose This draws the window. This is an override of DrawSelf() in LPane.

Access Virtual, Protected

Prototype
```
virtual void DrawSelf();
```

Parameters None

Return None

DrawSizeBox()

Purpose Draw standard size box for resizable Windows.

Access Virtual, Protected

Prototype
```
virtual void DrawSizeBox();
```

Parameters None

Return None

Enable()

Purpose Enable the Window. This is an override of Enable() in LPane.

Access Virtual, Public

Prototype
```
virtual void Enable();
```

Parameters None

Return None

EstablishPort()

Purpose Make Window the current Port.

Access Public

Prototype

```
virtual Boolean EstablishPort();
```

Parameters None

Return None

Remarks If you call this function directly, you should call LView::OutOfFocus() with `nil`, since changing the current port may invalidate the Focus.

ExpandSubPane()

Purpose Resize a subpane so that it is the same size as the Window, horizontally and/or vertically. This is an override of ExpandSubPane() in LView.

FetchWindowObject()

Purpose Return the PowerPlant Window object associated with a Mac OS `WindowPtr`.

Access Static, Public

Prototype

```
static LWindow* FetchWindowObject(
     WindowPtr inWindowP );
```

Parameters The parameter for this method is:

`WindowPtr`	`inWindowP`	The WindowPtr for the window.

Return Returns `nil` if the WindowPtr is not a PowerPlant Window.

FindCommandStatus()

Purpose Pass back whether a Command is enabled and/or marked (in a Menu). This is an override of FindCommandStatus() in LCommander.

FindWindowByID()

Purpose Return the Window object with the specified Pane ID number.

This method loops through all Windows from front to back searching for the first one with the specified Pane ID. Note that it's common to use the same PPob (and WIND) resource to create multiple windows, all of which have the same Pane ID. This function returns the frontmost match, whether it is visible or not.

Access Static, Public

Prototype

```
static LWindow* FindWindowByID(
     PaneIDT inWindowID );
```

Parameters The parameter for this method is:

`PaneIDT`	`inWindowID`	The pane ID for the Window.

Return Returns `nil` if no match found, else returns the pointer to the Window.

GetAEProperty()

Purpose This method returns an AppleEvent property.

Access Virtual, Public

Prototype

```
virtual void GetAEProperty(
     DescType inProperty,
     const AEDesc &inRequestedType,
     AEDesc &outPropertyDesc ) const;
```

Parameters The parameters for this method are:

`DescType`	`inProperty`	The descriptor type for the property.
`const AEDesc&`	`inRequestedType`	The AppleEvent descriptor type.
`AEDesc&`	`outPropertyDesc`	The property descriptor.

GetAEWindowAttribute()

Purpose Determines the Window attribute based on the received AppleEvent.

Access Public

Prototype

```
void GetAEWindowAttribute(
     EWindAttr inAttribute,
     AEDesc &outPropertyDesc ) const;
```

Parameters The parameters for this method are:

`EWindAttr`	`inAttribute`	The attributes for the Window.
`AEDesc&`	`outPropertyDesc`	The property descriptor.

Return None

GetDescriptor()

Purpose Return the title of a Window. This is an override of GetDescriptor() in LPane.

GetGlobalBounds()

Purpose Pass back the port rectangle of a Window in global coordinates.

Access Public

Prototype

```
void GetGlobalBounds(
     Rect &outBounds ) const;
```

Parameters The parameter for this method is:

`Rect&`	`outBounds`	The rectangle of the window in global coordinates.

Return None

GetMacPort()

Purpose Return Toolbox GrafPort associated with a Window object. This is an override of GetMacPort() in LPane.

GetMinMaxSize()

Purpose This method gets the value of mMinMaxSize.

Access Public

Prototype

```
void GetMinMaxSize( Rect &outRect) const;
```

Parameters The parameter for this method is:

`Rect&`	`outRect`	The rectangle of the window in global coordinates.

Return None

GetStandardSize()

Purpose This method returns the value of mStandardSize in the `outStdSize` parameter.

Access Public

Prototype
```
void GetStandardSize( SDimension16 &
     outStdSize ) const;
```

Parameters The parameter for this method is:

`SDimension16&`	`outStdSize`	The rectangle of the window in global coordinates.

Return None

GlobalToPortPoint()

Purpose Convert a point from global (screen) coordinates to a Window's Port coordinates. This is an override of GlobalToPortPoint() in LPane.

HandleAppleEvent()

Purpose This method handles an AppleEvent.

Access Virtual, Public

Prototype
```
virtual void HandleAppleEvent(
     const AppleEvent &inAppleEvent,
```

```
    AppleEvent &outAEReply,
    AEDesc &outResult,
    long inAENumber );
```

Parameters The parameters for this method are:

`const AppleEvent&`	`inAppleEvent`	The reference to the AppleEvent.
`AppleEvent&`	`outAEReply`	The AppleEvent reply.
`AEDesc&`	`outResult`	The AppleEvent result.
`long`	`inAENumber`	The AppleEvent number.

Return None

HandleClick()

Purpose Respond to a click on a Window.

The inPart parameter is the part code returned by the Toolbox FindWindow routine.

Access Virtual, Public

Prototype

```
virtual void HandleClick(
    const EventRecord &inMacEvent,
    Int16 inPart);
```

Parameters The parameters for this method are:

`const EventRecord&`	`inMacEvent`	The reference to the event information.
`Int16`	`inPart`	The part code returned by the Toolbox `FindWindow` routine.

Return None

HasAttribute()

Purpose Checks the Window to see if it has a given attribute.

Access Public

Prototype
```
Boolean HasAttribute( EWindAttr
     inAttribute ) const;
```

Parameters The parameter for this method is:

`EWindAttr`	`inAttribute`	The Window attribute to check for.

Return Returns `true` is the attribute is present, else `false`.

HideSelf()

Purpose Window is being made invisible. This is an override of HideSelf() in LPane.

Access Virtual, Protected

Prototype
```
virtual void HideSelf();
```

Parameters None

Return None

InitWindow()

Purpose Private Initializer from data in a struct.

Access Private

Prototype
```
void InitWindow(
     const SWindowInfo &inWindowInfo );
```

Parameters The parameter for this method is:

`const SWindowInfo&`	`inWindowInfo`	The Superwindow information.

Return None

InvalPortRect()

Purpose This method calls `InvalRect()` to invalidate the port rectangle. This is an override of InvalPortRect() in LPane.

InvalPortRgn()

Purpose This method calls `InvalRgn()`. This is an override of InvalPortRgn() in LPane.

MakeMacWindow()

Purpose Make a new Mac Window from a WIND resource template.

Access Private

Prototype

```
void MakeMacWindow( Int16 inWINDid );
```

Parameters The parameter for this method is:

`Int16`	`inWINDid`	The resource ID of the WIND template.

Return None

MakeSelfSpecifier()

Purpose AppleEvent Object Model Support

Access Virtual, Public

Prototype

```
virtual void MakeSelfSpecifier(
     AEDesc &inSuperSpecifier,
     AEDesc &outSelfSpecifier ) const;
```

Parameters The parameters for this method are:

`AEDesc&`	`inSuperSpecifier`	The AppleEvent super specifier.
`AEDesc&`	`outSelfSpecifier`	The AppleEvent self specifier.

Return None

MoveWindowBy()

Purpose Moves a Window.

Access Public

Prototype

```
void MoveWindowBy(
     Int16 inHorizDelta,
     Int16 inVertDelta );
```

Parameters The parameters for this method are:

`Int16`	`inHorizDelta`	The amount to move the window in the horizontal directions.
`Int16`	`inVertDelta`	The amount to move the window in the vertical directions.

Return None

MoveWindowTo()

Purpose Move the top left corner of the Window's port rect to the specified location in Global coordinates.

Access Public

Prototype

```
void MoveWindowTo(
     Int16 inHoriz,
     Int16 inVert );
```

Parameters The parameters for this method are:

`Int16`	`inHoriz`	The horizontal place to move the window to.
`Int16`	`inVert`	The vertical place to move the window to.

Return None

ObeyCommand()

Purpose Handle a command. This is an override of ObeyCommand() in LCommander.

PortToGlobalPoint()

Purpose Convert a point from Port to Global (screen) coordinates. Refer to the discussion for GlobalToPortPoint() for more comments. This is an override of PortToGlobalPoint() in LPane.

ResizeFrameBy()

Purpose Change the Frame size by the specified amounts. This is an override of ResizeFrameBy() in LPane.

ResizeWindowBy()

Purpose Change the size of a Window by the specified number of pixels.

Access Public

Prototype

```
void ResizeWindowBy(
     Int16 inWidthDelta,
     Int16 inHeightDelta );
```

Parameters The parameters for this method are:

`Int16`	`inWidthDelta`	The horizontal width change to resize the window by.
`Int16`	`inHeightDelta`	The vertical height change to resize the window by.

Return None

ResizeWindowTo()

Purpose Change the size of a Window to the specified width and height (in pixels).

Access Public

Prototype

```
void ResizeWindowTo( Int16 inWidth, Int16 inHeight
     );
```

Parameters The parameters for this method are:

`Int16`	`inWidth`	The horizontal width for the Window.
`Int16`	`inHeight`	The vertical height for the Window.

Return None

Resume()

Purpose Brings the window to the front.

Access Virtual, Public

Prototype

```
virtual void Resume();
```

Parameters None

Return None

Select()

Purpose Selects the Window.

Access Virtual, Public

Prototype
```
virtual void Select();
```

Parameters None

Return None

SendAESetBounds()

Purpose AppleEvent for changing the size of a Window.

inBounds is the new port rectangle for the Window, in Global coordinates. Set inExecuteAE to true to actually resize the Window, false to just send the event for script recording purposes.

Access Public

Prototype
```
virtual void SendAESetBounds(
     Rect *inBounds,
     Boolean inExecuteAE );
```

Parameters The parameters for this method are:

`Rect*`	`inBounds`	The new port rectangle for the Window, in Global coordinates.
`Boolean`	`inExecuteAE`	Set true to actually resize the Window, false to just send the event for script recording purposes.

Return None

SendAESetPosition()

Purpose AppleEvent for moving a Window to a new position.

Access Public

Prototype

```
virtual void SendAESetPosition(
    Point inPosition,
    Boolean inExecuteAE);
```

Parameters The parameters for this method are:

`Point`	`inPosition`	The location for the top left corner of the Window's port rectangle, in Global coordinates.
`Boolean`	`inExecuteAE`	Set true to actually resize the Window, false to just send the event for script recording purposes.

Return None

Remarks You'll use the `false` value for `inExecuteAE` when you have already moved the window in response to tracking user actions (the Toolbox trap `DragWindow` moves the Window).

SendAESetZoom()

Purpose AppleEvent for zooming a Window.

This method figures out whether to zoom in or out based on the current window size and location

Access Public

Prototype

```
virtual void SendAESetZoom();
```

Parameters None

Return None

SetAEProperty()

Purpose Set the AppleEvent property.

Access Virtual, Public

Prototype

```
virtual void SetAEProperty(
     DescType inProperty,
     const AEDesc &inValue,
     AEDesc& outAEReply );
```

Parameters The parameters for this method are:

`DescType`	`inProperty`	The property descriptor.
`const AEDesc&`	`inValue`	The AppleEvent descriptor.
`AEDesc&`	`outAEReply`	The AppleEvent reply.

Return None

SetAttribute()

Purpose Sets an attribute for the Window.

Access Public

Prototype

```
void SetAttribute( EWindAttr inAttribute );
```

Parameters The parameter for this method is:

`EWindAttr`	`inAttribute`	The attribute for the Window.

Return None

SetDescriptor()

Purpose Set the title of a Window. This is an override of SetDescriptor() in LPane.

SetForeAndBackColors()

Purpose Specify the foreground and/or background colors of a Window. This is an override of SetForeAndBackColors() in LPane.

SetMinMaxSize()

Purpose This is an accessor method for mMinMaxSize.

Access Public

Prototype

```
void SetMinMaxSize( const Rect &inRect );
```

Parameters The parameter for this method is:

`const Rect&`	`inRect`	The Rect to set mMinMaxSize to.

Return None

SetStandardSize()

Purpose Set the standard size (mStandardSize) for the Window.

Access Public

Prototype

```
void SetStandardSize( SDimension16 inStdSize );
```

Parameters The parameter for this method is:

`SDimension16`	`inStdSize`	The Window's standard size (set the value of mStandardSize from this).

Return None

Show()

Purpose Make a Window visible. This is an override of Show() in LPane.

ShowSelf()

Purpose Window is being made visible. This is an override of ShowSelf() in LPane.

Suspend()

Purpose This method hides the window if the `windAttr_HideOnSuspend` attribute is present, else it deactivates the window if it is active and suspend occurs.

Access Virtual, Public

Prototype
```
virtual void Suspend();
```

Parameters None

Return None

UpdatePort()

Purpose Redraw invalidated area of the Window. The Mac WindowPtr maintains an update region that defines the area that needs to be redrawn. This is an override of UpdatePort() in LPane.

ValidPortRect()

Purpose This method calls ValidRect(). This is an override of ValidPortRect() in LPane.

ValidPortRgn()

Purpose This method calls ValidRect(). This is an override of ValidPortRgn() in LPane.

mMacWindowP

Purpose This stores the Mac OS `WindowPtr`.

Access Protected

Prototype

```
WindowPtr mMacWindowP;
```

mMinMaxSize

Purpose This stores the min and max size for the Window.

Access Protected

Prototype

```
Rect mMinMaxSize;
```

mStandardSize

Purpose This stores the standard size for the Window.

Access Protected

Prototype

```
SDimension16 mStandardSize;
```

mUserBounds

Purpose This stores the user bounds Rect for the Window.

Access Protected

Prototype

```
Rect mUserBounds;
```

mAttributes

Purpose This stores the attributes for the Window.

Access Protected

Prototype
```
Uint32 mAttributes;
```

mForeColor

Purpose This stores the foreground color for the Window.

Access Protected

Prototype
```
RGBColor mForeColor;
```

mBackColor

Purpose This stores the background color for the Window.

Access Protected

Prototype
```
RGBColor mBackColor;
```

mMoveOnlyUserZoom

Purpose This stores information about zooming a Window that is at standard size, but is partially offscreen.

Access Protected

Prototype
```
Boolean mMoveOnlyUserZoom;
```

LYieldAttachment

Description LYieldAttachment is a PowerPlant class that is used for yielding control to another thread before returning. It obviates the need to override LApplication`::ProcessNextEvent()` in order to give time to other threads.

Methods The methods in this class are:

LYieldAttachment() ExecuteSelf()

Data Members The data members in this class are:

mQuantum mNextTicks

Operation The concept of an Attachment is a powerful underlying concept of PowerPlant's inner workings. For a detailed discussion on this topic, refer to *The PowerPlant Book*.

Source files (Threads Classes)

`UThread.h`

`UThread.cp`

See also LAttachment

LThread

LYieldAttachment()

Purpose The constructors create the object and initialize the data members to the passed-in values.

Access Public

Prototype

```
LYieldAttachment( SInt32 inQuantum);
```

Parameters The optional parameter to the constructor lets you specify how much time should elapse before control returns to the caller. This is useful for preventing `WaitNextEvent()` from being called too often.

ExecuteSelf()

Purpose This method yields control to other threads. Note that control is always yielded at least once.

Access Virtual, Public

Prototype

```
void ExecuteSelf(MessageT inMessage,
     void* ioParam );
```

Parameters This method does not use its parameters.

Return None

mQuantum

Purpose The quantum, or wait time in system ticks between calls to LThread::Yield().

Access Protected

Prototype

```
SInt32 mQuantum;
```

mNextTicks

Purpose A private value used to decide when to yield.

Access Protected

Prototype

```
UInt32 mNextTicks;
```

StAsyncOperation

Overview

`StAsyncOperation` is a PowerPlant helper class used by the networking classes to delay thread execution while an asynchronous operation executes.

Methods

The methods in this class are:

StAsyncOperation()	~StAsyncOperation()
AbortOperation()	GetThreadOperation()
Int_AsyncResume()	WaitForResult()

Data Members

The data members in this class are:

mThread	mResult

Source files

(Networking Classes)

`UNetworking.h`

`UNetworking.cp`

See also

LInterruptSafeList

LMacTCPDNSOperation

StMacTCPOperation

StMacTCPUDPOperation

StOpenTptOperation

StAsyncOperation()

Purpose

The constructor creates the object.

Access

Public

Prototype

```
StAsyncOperation();
```

~StAsyncOperation()

Purpose The destructor destroys the object.

Access Public

Prototype
```
~StAsyncOperation();
```

AbortOperation()

Purpose Abort the operation.

Access Virtual, Public

Prototype
```
virtual void AbortOperation();
```

Parameters None

Return None

GetThreadOperation()

Purpose Return the operation object.

Access Static, Public

Prototype
```
static StAsyncOperation *GetThreadOperation(
     LThread *inThread );
```

Parameters This method has the following parameter:

`LThread*`	`inThread`	The thread.

Return The operation object.

Int_AsyncResume()

Purpose This routine might be called at interrupt time

IMPORTANT: If you override this operation, be sure to override AbortOperation() as well if necessary.

Access Virtual, Public

Prototype
```
virtual void Int_AsyncResume( OSStatusinResult );
```

Parameters This method has the following parameter:

`OSStatus`	`inResult`	The status code.

Return None

WaitForResult()

Purpose Wait for the result by blocking this thread.

Access Public

Prototype
```
void WaitForResult();
```

Parameters None

Return None

mThread

Purpose Storage for the thread.

Access Protected

Prototype
```
LThread* mThread;
```

mResult

Purpose The storage for the status code.

Access Protected

Prototype `OSStatus mResult;`

sPendingOperations

Purpose A list of the pending operations.

Access static Protected

Prototype `static LInterruptSafeList *sPendingOperations;`

StCritical

Description StCritical is a PowerPlant wrapper class for working with critical sections. It would be useful as a stack object.

Methods The methods in this class are:

StCritical() ~StCritical()

Data Members None

Source files (Threads Classes)

`UThread.h`

`UThread.cp`

See Also LThread

StCritical()

Purpose Wrapper constructor which calls `ThreadBeginCritical()` to signal the entry of a critical section.

Access Public

Prototypes

```
StCritical();
```

Parameters None

~StCritical()

Purpose Wrapper destructor which calls `ThreadEndCritical()` to signal the exit of a critical section.

Access Public

Prototype

```
~StCritical();
```

StCursor

Overview StCursor is a PowerPlant class that temporarily changes the cursor to something else.

Methods The methods in this class are:

StCursor() ~StCursor()

Data Members The data members in this class are:

mRestoreID

Operation Upon entry, specify the ResIDT of the 'CURS' to change to. Upon exit, we either restore the old cursor or default to the arrow.

By default, StCursor will set to the watch cursor, and restore the original cursor.

Source files (Utility Classes)

`UCursor.h`

`UCursor.cp`

StCursor()

Purpose The default constructor initializes the object.

Access Public

Prototype

```
StCursor(ResIDTinCursID = watchCursor,
     BooleaninRestoreOriginal = true);
```

StCursor(const StCursor &inOriginal);

```
StCursor&operator = (const StCursor &inOriginal);
```

Parameters The cursor resource ID and whether to restore the cursor or not. The assignment operator and copy constructor require a reference to the original cursor object.

~StCursor()

Purpose The destructor destroys the object.

Access Public, Virtual

Prototype
```
virtual ~StCursor();
```

Parameters None

mRestoreID

Purpose The resource ID for restoring.

Access Protected

Prototype ResIDT mRestoreID;

StDialogHandler

Overview

StDialogHandler is a PowerPlant class that is used for creating dialog boxes.

Methods

The methods in this class are:

StDialogHandler() ~StDialogHandler()

AllowSubRemoval() DoDialog()

FindCommandStatus() GetDialog()

ListenToMessage() SetSleepTime()

Data Members

The data members in this class are:

mDialog mMessage

mSleepTime

Operation

Refer to *The PowerPlant Book* for important information about mixing the PowerPlant dialog classes with the Mac OS Toolbox calls.

Source files

(Utility Classes)

`UModalDialogs.h`

`UModalDialogs.cp`

Ancestors

LAttachable

LCommander

LEventDispatcher

LListener

See also

LBroadcaster

StDialogHandler()

Purpose The constructor creates an object from the passed-in parameters.

Access Public

Prototype

```
StDialogHandler( ResIDT inDialogResID,
     LCommander *inSuper );
```

Parameters The parameters for this constructor are:

`ResIDT`	`inDialogResID`	The resource ID for the dialog resource.
`LCommander*`	`inSuper`	The SuperCommander for this dialog.

~StDialogHandler()

Purpose The destructor destroys the object.

Access Virtual, Public

Prototype

```
virtual ~StDialogHandler();
```

AllowSubRemoval()

Purpose This method indicates whether removal of subcommanders is allowed. It is an override of AllowSubRemoval() in LCommander.

DoDialog()

Purpose Handle an event for a dialog box.

Access Virtual, Public

Prototype

```
virtual MessageT DoDialog();
```

Parameters None

Return If the event triggers a Broadcaster to broadcast a message, the last such message heard by the DialogHandler is returned. Otherwise, this function returns `msg_Nothing`.

Remarks Call this function repeatedly to handle events.

FindCommandStatus()

Purpose This method passes back the status of a command. It is an override of FindCommandStatus() in LCommander.

GetDialog()

Purpose This is an accessor method for mDialog.

Access Public

Prototype

```
LWindow* GetDialog();
```

Parameters None

Return Returns the pointer mDialog to the window for the dialog box.

ListenToMessage()

Purpose This method provides the message processing behavior for an LListener-inherited object. It is an override of ListenToMessage() in LListener.

SetSleepTime()

Purpose This method sets the value of the mSleepTime data member.

Access Public

Prototype

```
void SetSleepTime(Int32 inSleepTime);
```

Parameters Int32 indicating the sleep time to store in mSleepTime.

Return None

mDialog

Purpose The pointer to the dialog box.

Access Protected

Prototype
```
LWindow *mDialog;
```

mMessage

Purpose A place to store the message.

Access Protected

Prototype
```
MessageT mMessage;
```

mSleepTime

Purpose Storage for the sleep time parameter.

Access Protected

Prototype
```
Int32 mSleepTime;
```

StMacTCPOperation

Overview StMacTCPOperation is a PowerPlant class that is used for managing TCPParamBlock and thread blocking for MacTCP interface calls. This keeps a global reference to the MacTCP driver and DNS code segment.

Methods The methods in this class are:

StMacTCPOperation() ~StMacTCPOperation()

AsyncRun() GetCompletionProc()

GetParamBlock() Int_TCPCompletionProc()

Run()

Data Members The data members in this class are:

mTCPParamBlock sMacTCPCompletionProc

sTCPParamBlockDeleteQueue

Source files (Networking Classes)

`UMacTCPSupport.h`

`UMacTCPSupport.cp`

See also StAsyncOperation

StMacTCPListenOperation

StMacTCPSendOperation

StMacTCPOperation()

Purpose The constructor creates the object.

Access Public

Prototype

```
StMacTCPOperation( Int16 inOperationCode,
```

```
    StreamPtr inStreamPtr );
```

Parameters This constructor has the following parameters:

`Int16`	`inOperationCode`	The operation code.
`StreamPtr`	`inStreamPtr`	The stream pointer.

~StMacTCPOperation()

Purpose The destructor destroys the object.

Access Public

Prototype

```
~StMacTCPOperation();
```

AsyncRun()

Purpose Run the operation and return immediately. The completion will appear as a notification on the endpoint.

Functions using this method can NOT be aborted.

Presently used only for LMacTCPTCPEndpoint::SendDisconnect() to mimic behavior of OT.

Access Public

Prototype

```
void AsyncRun( LMacTCPTCPEndpoint* inEndpoint,
     Uint16 inEventCode );
```

Parameters This method has the following parameters:

`LMacTCPTCPEndpoint*`	`inEndpoint`	The endpoint.
`Uint16`	`inEventCode`	The event code.

Return None

GetCompletionProc()

Purpose Return the value of sMacTCPCompletionProc.

Access Inline, Public

Prototype
```
inline TCPIOCompletionUPP GetCompletionProc();
```

Parameters None

Return The Completion routine Universal Procedure Pointer (UPP).

GetParamBlock()

Purpose Retrieve the value of mTCPParamBlock.

Access Inline, Public

Prototype
```
inline STCPParamBlock&GetParamBlock();
```

Parameters None

Return The parameter block.

Int_TCPCompletionProc()

Purpose The completion routine for the operation.

Access Protected, Static

Prototype
```
pascal void Int_TCPCompletionProc(
     STCPParamBlock*inParamBlock
       #if !GENERATINGCFM
         : __A0
       #endif
   );
```

Parameters This method has the following parameters:

`STCPParamBlock*`	`inParamBlock`	The endpoint.

Return None

Run()

Purpose Attempt the operation. Check for immediate failure.

Access Public

Prototype
```
void Run();
```

Parameters None

Return None

mTCPParamBlock

Purpose The TCP parameter block.

Access Protected

Prototype
```
STCPParamBlock * mTCPParamBlock;
```

sMacTCPCompletionProc

Purpose The MacTCP completion procedure.

Access Private, Static

Prototype
```
TCPIOCompletionUPP sMacTCPCompletionProc;
```

sTCPParamBlockDeleteQueue

Purpose The MacTCP parameter block deletion queue.

Access Private, Static

Prototype LTCPParamBlockDeleteQueue * sTCPParamBlockDeleteQueue;

StMacTCPUDPOperation

Overview

StMacTCPUDPOperation is a PowerPlant class that is used for managing UDPParamBlock and thread blocking for MacTCP related UDP interface calls.

Methods

The methods in this class are:

StMacTCPUDPOperation() ~StMacTCPUDPOperation()

GetCompletionProc() GetParamBlock()

Int_UDPCompletionProc() Run()

Data Members

The data members in this class are:

mUDPParamBlock sMacTCPUDPCompletionProc

sUDPParamBlockDeleteQueue

Source files

(Networking Classes)

`UMacTCPSupport.h`

`UMacTCPSupport.cp`

See also

LGlobalsContext

StAsyncOperation

StMacTCPUDPSendOperation

StMacTCPUDPOperation()

Purpose

The constructor creates the object.

Access

Public

Prototype
```
StMacTCPUDPOperation(
     Int16 inOperationCode,
     StreamPtr inStreamPtr );
```

Parameters None

~StMacTCPUDPOperation()

Purpose The destructor destroys the object.

Access Public

Prototype
```
~StMacTCPUDPOperation();
```

GetCompletionProc()

Purpose Return the value of sMacTCPUDPCompletionProc.

Access Inline, Public

Prototype
```
inline UDPIOCompletionUPP GetCompletionProc();
```

Parameters None

Return The procedure pointer.

GetParamBlock()

Purpose Return the value of mUDPParamBlock.

Access Inline, Public

Prototype
```
inline SUDPParamBlock&GetParamBlock();
```

Parameters None

Return The parameter block.

Int_UDPCompletionProc()

Purpose The completion procedure.

Access Static, Protected

Prototype

```
static pascal void Int_UDPCompletionProc(
     SUDPParamBlock* inParamBlock
       #if !GENERATINGCFM
         : __A0
       #endif
   );
```

Parameters This method has the following parameters:

`SUDPParamBlock*`	`inParamBlock`	The parameter block.

Return None

Run()

Purpose Attempt the operation. Check for immediate failure.

Access Public

Prototype

```
void Run();
```

Parameters None

Return None

mUDPParamBlock

Purpose The parameter block.

Access Protected

Prototype

```
SUDPParamBlock * mUDPParamBlock;
```

sMacTCPUDPCompletionProc

Purpose The completion routine Universal Procedure Pointer (UPP).

Access Static, Private

Prototype
```
UDPIOCompletionUPP sMacTCPUDPCompletionProc;
```

sUDPParamBlockDeleteQueue

Purpose The parameter block deletion queue.

Access Static, Private

Prototype
```
LUDPParamBlockDeleteQueue
     *sUDPParamBlockDeleteQueue;
```

StMutex

Description StMutex is a PowerPlant wrapper class for working with mutual exclusions (mutexes).

Methods The methods in this class are:

StMutex() ~StMutex()

Data Members mMutex

Source files (Threads Classes)

`UThread.h`

`UThread.cp`

See Also LMutexSemaphore

LThread

StMutex()

Purpose Constructor waits on the given mutual exclusion semaphore.

Access Public

Prototypes

```
StMutex(LMutexSemaphore& inMutex);
```

Parameters A reference to the mutex to wait for.

~StMutex()

Purpose Destructor releases the semaphore.

Access Public

Prototype

```
~StMutex();
```

mMutex

Purpose The mutex to wait for.

Access Private

Prototype

```
LMutexSemaphore&mMutex;
```

StOpenTptOperation

Overview StOpenTptOperation is a PowerPlant class that is used for encapsulating an Open Transport operation. It manages thread blocking for OpenTransport interface calls.

Methods The methods in this class are:

StOpenTptOperation()	~StOpenTptOperation()
GetCookie()	GetEventCode()
GetResultCode()	Int_TimerProc()
OperationTimedOut()	SetEventCode()
WaitForCompletion()	WaitForResult()

Data Members The data members in this class are:

mNotifHandler	mEventCode
mCookieTest	mTestCookie
mResultCode	mCookie
mOperationTimeout	sOTOpTimerUPP

Source files (Networking Classes)

`UOpenTptOperation.h`

`UOpenTptOperation.cp`

See also LGlobalsContext

StOpenTptOperation()

Purpose The constructor creates the object.

Access Public

Prototype

```
StOpenTptOperation(
```

```
LOpenTptNotifHandler * inNotifHandler,
OTEventCode inEventCode,
void * inCookieTest,
Boolean inTestCookie );
```

Parameters This constructor has the following parameter:

`LOpenTptNotifHandler *`	`inNotifHandler`	This is the Open Transport notification handler.
`OTEventCode`	`inEventCode`	The event code.
`void*`	`inCookieTest`	The cookie test value.
`Boolean`	`inTestCookie`	Whether to test the cookie.

~StOpenTptOperation()

Purpose The destructor destroys the object.

Access Public

Prototype

```
~StOpenTptOperation();
```

GetCookie()

Purpose Returns the value of mCookie.

Access Public

Prototype

```
void* GetCookie();
```

Parameters None

Return The value of mCookie.

GetEventCode()

Purpose Return the value of mEventCode.

Access Public

Prototype
```
OTEventCode GetEventCode();
```

Parameters None

Return The value of mEventCode.

GetResultCode()

Purpose Return the value of mResultCode.

Access Public

Prototype
```
OTResult GetResultCode();
```

Parameters None

Return The value of mResultCode.

Int_TimerProc()

Purpose Install a Time Manager proc.

Access Static, Protected

Prototype
```
pascal void Int_TimerProc(TMTaskPtr tmTaskPtr
     #if !GENERATINGCFM
     : __A1
     #endif);
```

Parameters The parameter is the pointer to a Time Manager task block.

Return None

OperationTimedOut()

Purpose Return the value of mOperationTimeout.

Access Public

Prototype

```
Boolean OperationTimedOut();
```

Parameters None

Return The value of mOperationTimeout.

SetEventCode()

Purpose Set the value of mEventCode.

Access Public

Prototype

```
void SetEventCode(OTEventCode inEventCode);
```

Parameters This method has the following parameter:

`OTEventCode`	`inEventCode`	The Open Transport event code.

Return None

WaitForCompletion()

Purpose Wait for the Time Manager task to complete.

Access Public

Prototype

```
void WaitForCompletion( Uint32 inTimeoutSeconds);
```

Parameters This method has the following parameter:

`Uint32`	`inTimeoutSeconds`	The Open Transport event code.

Return None

WaitForResult()

Purpose This method breaks into the debugger with a call to `DebugStr()`.

Access Public

Prototype
```
void WaitForResult();
```

Parameters None

Return None

mNotifHandler

Purpose The notification handler.

Access Protected

Prototype
```
LOpenTptNotifHandler * mNotifHandler;
```

mEventCode

Purpose The event code.

Access Protected

Prototype
```
OTEventCode mEventCode;
```

mCookieTest

Purpose The test cookie.

Access Protected

Prototype `void * mCookieTest;`

mTestCookie

Purpose Whether to test the cookie.

Access Protected

Prototype `Boolean mTestCookie;`

mResultCode

Purpose The result code.

Access Protected

Prototype `OTResult mResultCode;`

mCookie

Purpose The cookie pointer.

Access Protected

Prototype `void* mCookie;`

mOperationTimeout

Purpose Operation timed out.

Access Protected

Prototype `Boolean mOperationTimeout;`

sOTOpTimerUPP

Purpose The Open Transport operation timer proc pointer.edure

Access Static, Protected

Prototype TimerUPP sOTOpTimerUPP;

StSetupGlobals

Overview StSetupGlobals is a PowerPlant class that is used for saving and restoring the global variables context.

Methods The methods in this class are:

StSetupGlobals() ~StSetupGlobals()

Data Members The data members in this class are:

mOldGlobals

Source files (Networking Classes)

`UCallbackUtils.h`

`UCallbackUtils.cp`

See also LGlobalsContext

StSetupGlobals()

Purpose For 68K, the constructor sets the A5 (or A4) to the value stored in the given LGlobalsContext object. Destructor restores A5/A4 to its previous value.

For PowerPC or 68K Code Fragment Manager, there is no need to set up A5/A4/RTOC since the Mixed Mode Manager will do this automatically. The constructor and destructor do nothing in this case.

Access Inline, Public

Prototype

```
inline StSetupGlobals(
     LGlobalsContext& inGlobalsContext );
```

Parameters This constructor has the following parameter:

`LGlobalsContext&`	`inGlobals Context`	This is a reference to the globals context.

~StSetupGlobals()

Purpose For 68K, the destructor calls `SetA4()` or `SetA5()`.

For PowerPC or 68K Code Fragment Manager, there is no need to restore A5/A4/RTOC since the Mixed Mode Manager will do this automatically. The constructor and destructor do nothing in this case.

Access Inline, Public

Prototype

```
inline ~StSetupGlobals();
```

Parameters None

mOldGlobals

Purpose Storage for the old globals.

Access Private

Prototype

```
long mOldGlobals;
```

TArray

Overview TArray is a PowerPlant template class that is a template wrapper for LArray.

Methods The methods in this class are:

TArray()	~TArray()
AddItem()	AssignItemsAt()
FetchIndexOf()	FetchInsertIndexOf()
FetchItemAt()	FetchItemPtr()
InsertItemsAt()	Remove()
Operator[ ]	

Data Members There are no data members in this class.

Operation You can't store actual objects in a TArray or LArray. Only use TArray and LArray to store pointers to objects (created via new), built-in numerical data types, or data structures (simple structs).

Source files (Array Classes)

TArray.h

Ancestors LArray

See Also LComparator

TArray()

Purpose Construct an array.

Access Public

Prototype

```
TArray( LComparator *inComparator = nil,
     Boolean inKeepSorted = false );
```

```
TArray( UInt32 inItemCount,
     LComparator *inComparator = nil,
     Boolean inKeepSorted = false );
TArray( Handle inItemsHandle,
     LComparator *inComparator = nil,
     Boolean inIsSorted = false,
     Boolean inKeepSorted = false );
```

Parameters The parameters for these constructors are:

`LComparator*`	`inComparator`	A pointer to the comparator to use for the array.
`Boolean`	`inKeepSorted`	A value indicating whether the array should be kept sorted.
`UInt32`	`inItemCount`	The number of items to put in the array.
`Handle`	`inItemsHandle`	A handle to the items to put in the array.
`Boolean`	`inIsSorted`	A value indicating whether the items are sorted.

~TArray()

Purpose The destructor destroys the array.

Access Virtual, Public

Prototype

```
virtual ~TArray();
```

AddItem()

Purpose This adds an item to the array. This is an override of AddItem() in LArray.

AssignItemsAt()

Purpose Assign the same value to items in the array starting at the specified index. This is an override of AssignItemsAt() in LArray.

FetchIndexOf()

Purpose Returns the index of the specified item within the array. This is an override of FetchIndexOf() in LArray.

FetchInsertIndexOf()

Purpose Return the index at which the specified item would be inserted. This is an override of FetchInsertIndexOf() in LArray.

FetchItemAt()

Purpose Pass back the item at the specified index. This is an override of FetchItemAt() in LArray.

FetchItemPtr()

Purpose Returns a pointer to an item.

Access Public

Prototype

```
T* FetchItemPtr( ArrayIndexT inAtIndex ) const;
```

Parameters The parameter for this method is:

`ArrayIndexT`	`inAtIndex`	The array index to fetch from.

Return A pointer to the array item at the specified index.

InsertItemsAt()

Purpose Insert items at the specified position in an array. This is an override of InsertItemsAt() in LArray.

Remove()

Purpose Remove an item from an array. This is an override of Remove() in LArray.

Operator[ ]

Purpose The array operator is overloaded for this class to provide convenient array-like access to array items, even though they are implemented differently than the row/column arrangement in memory.

Access Public

Prototype

```
T& operator [] ( ArrayIndexT inAtIndex );
const T& operator [] (ArrayIndexT inAtIndex) const;
```

Parameters The parameter for this method is:

`ArrayIndexT`	`inAtIndex`	The array index to access.

Return A reference to the array element/item of interest.

UCursor

Overview

UCursor is a PowerPlant class that wraps calls to the Toolbox routine named `SetTheCursor()`.

Methods

The methods in this class are:

GetCurrentID()
InitTheCursor()
InAnimatedCursor()
SetArrow()
SetCurrentID()
SetCross()
SetInAnimatedCursor()
SetIBeam()
SetPlus()
SetTheCursor()
SetWatch()

Data Members

The data members in this class are:

sCurrentID
sInAnimatedCursor

Operation

Given the ResIDT for a 'CURS' resource, set the cursor to that. If the cursor is currently hidden, `SetTheCursor()` will not make it visible. There is no Apple-supported or recommend way to determine cursor visibility.

You could refer to the "develop" Q&A section from the June 1997 issue of MacTech Magazine for information on how to determine cursor visibility, but use this at your own risk.

Source files

(Utility Classes)

`UCursor.h`

`UCursor.cp`

See Also

StCursor

GetCurrentID()

Purpose Retrieves the value of sCurrentID.

Access Static, Public, Inline

Prototype
```
staticResIDTGetCurrentID(){ return sCurrentID; }
```

Parameters None

InitTheCursor()

Purpose Cursor Initialization.

Access Static, Public

Prototype
```
staticvoidInitTheCursor();
```

Parameters None

InAnimatedCursor()

Purpose Returns the value of sInAnimatedCursor.

Access Static, Public, Inline

Prototype
```
staticBooleanInAnimatedCursor(){
     return sInAnimatedCursor; }
```

Parameters None

SetArrow()

Purpose Sets the cursor to an arrow.

Access Static, Public, Inline

Prototype
```
staticvoidSetArrow(){ SetTheCursor(0); }
```

Parameters None

SetCurrentID()

Purpose Sets the cursor ID.

Access Static, Public, Inline

Prototype

```
staticvoid SetCurrentID(const ResIDT inCurrentID)
     { sCurrentID = inCurrentID; }
```

Parameters The resource ID to set.

SetCross()

Purpose Sets the cursor to a cross.

Access Static, Inline, Public

Prototype

```
staticvoidSetCross(){ SetTheCursor(crossCursor); }
```

Parameters None

Return None

SetInAnimatedCursor()

Purpose Sets the value of sInAnimatedCursor.

Access Static, Inline, Public

Prototype

```
static   void SetInAnimatedCursor(Boolean inInAnimated)
         { sInAnimatedCursor = inInAnimated; }
```

Parameters Boolean value to set.

Return None

SetIBeam()

Purpose Sets the cursor to an I-beam.

Access Static, Inline, Public

Prototype static void SetIBeam(){ SetTheCursor(iBeamCursor); }

Parameters None

Return None

SetPlus()

Purpose Sets the cursor to a plus sign (+).

Access Static, Inline, Public

Prototype
```
staticvoidSetPlus(){ SetTheCursor(plusCursor); }
```

Parameters None

Return None.

SetTheCursor()

Purpose Sets the cursor to a value.

Access Static, Public

Prototype
```
staticvoidSetTheCursor(ResIDT inCursID);
```

Parameters The cursor resource ID.

SetWatch()

Purpose Sets the cursor to a watch.

Access Static, Inline, Public

Prototype `staticvoidSetWatch(){ SetTheCursor(watchCursor); }`

Parameters None

Return None.

sCurrentID

Purpose Current resource ID.

Access Protected

Prototype `staticResIDTsCurrentID;`

sInAnimatedCursor

Purpose Cursor animation.

Access Protected

Prototype `staticBooleansInAnimatedCursor;`

UDNSCache

Overview UDNSCache is a PowerPlant class that is used for a local cache for DNS entries.

Methods The methods in this class are:

AddToDNSCache() CheckCache()

CreateDNSCacheElem() GetAddressFromCache()

GetNameFromCache()

Data Members The data members in this class are:

sOTDNSNameCache sOTDNSAddressCache

Source files (Networking Classes)

`UDNSCache.h`

`UDNSCache.cp`

AddToDNSCache()

Purpose Add an entry to the DNS cache.

Access Static, Public

Prototype

```
static void AddToDNSCache( Uint32 inHostIP,
    ConstStringPtr inHostName);
```

Parameters The parameters for this method are:

`UInt32`	`inHostIP`	The host IP address.
`ConstStringPtr`	`inHostName`	The host name string.

Return None

CheckCache()

Purpose Verify the cache.

Access Static, Public

Prototype

```
static void CheckCache();
```

Parameters None

Return None

CreateDNSCacheElem()

Purpose Create a DNS cache element.

Access Static, Protected

Prototype

```
static void CreateDNSCacheElem(Uint32 inHostIP,
    ConstStringPtr inHostName,
    SDNSCacheElem& outElem );
```

Parameters The parameters for this method are:

`UInt32`	`inHostIP`	The host IP address.
`ConstStringPtr`	`inHostName`	The host name string.
`SDNSCacheElem&`	`outElem`	The element.

Return None

GetAddressFromCache()

Purpose Retrieve a DNS entry from the cache.

Access Static, Public

Prototype

```
static Boolean GetAddressFromCache(
```

```
Uint32 inHostIP,
LStr255& outHostName );
```

Parameters The parameters for this method are:

`Uint32`	`inHostIP`	The host IP address.
`LStr255&`	`outHostName`	The host name string.

Return None

GetNameFromCache()

Purpose Retrieve a DNS name from the cache.

Access Static, Public

Prototype
```
static Uint32 GetNameFromCache(ConstStringPtr
inHostName);
```

Parameters The parameters for this method are:

`ConstStringPtr`	`inHostName`	The host name string.

Return Returns the IP address if found in the cache. If not found, returns zero.

sOTDNSNameCache

Purpose The name cache array.

Access Static, Private

Prototype
```
static LArray* sOTDNSNameCache;
```

sOTDNSAddressCache

Purpose The address cache array.

Access Static, Private

Prototype

```
static LArray* sOTDNSAddressCache;
```

UMacTCPSupport

Overview UMacTCPSupport is a PowerPlant class that is used for helpers for the MacTCP classes. Keeps global reference to MacTCP driver and DNS code segment.

Methods The methods in this class are:

GetMacTCPRefNum() HasMacTCP()

OpenMacTCPDriver()

Data Members The data members in this class are:

sMacTCPRefNum sCloseResolverTask

Source files (Networking Classes)

`UMacTCPSupport.h`

`UMacTCPSupport.cp`

GetMacTCPRefNum()

Purpose Return the MacTCP reference number, held in sMacTCPRefNum.

Access Static, Public

Prototype

```
static Int16 GetMacTCPRefNum();
```

HasMacTCP()

Purpose Determine whether MacTCP is present.

Access Static, Public

Prototype

```
static Boolean HasMacTCP();
```

Parameters None

Return Returns true of MacTCP is present, false otherwise.

OpenMacTCPDriver()

Purpose Open the MacTCP driver if not already open.

Access Static, Public

Prototype
```
static void OpenMacTCPDriver();
```

Parameters None

Return None

sMacTCPRefNum

Purpose The MacTCP reference number.

Access Static, Private

Prototype
```
static Int16 sMacTCPRefNum;
```

sCloseResolverTask

Purpose Storage for the LMacTCP_CloseResolver.

Access Static, Private

Prototype
```
static LMacTCP_CloseResolver* sCloseResolverTask;
```

UNetworkFactory

Overview UNetworkFactory is a PowerPlant class that is used for easily creating TCP endpoints and mappers based on the current running system software.

Methods The methods in this class are:

CreateInternetMapper() CreateTCPEndpoint()

CreateUDPEndpoint() HasMacTCP()

HasOpenTransport() HasTCP()

Data Members There are no data members in this class.

Operation Calling simple methods in this utility class automatically chooses MacTCP or OpenTransport support transparently.

Source files (Networking Classes)

`UNetworkFactory.h`

`UNetworkFactory.cp`

See also LEndpoint

LTCPEndpoint

LUDPEndpoint

CreateInternetMapper()

Purpose Create the best name mapper object for Internet address resolution given the available system software.

Access Static, Public

Prototype

```
LInternetMapper* CreateInternetMapper();
```

Parameters None

Return A pointer to the mapper object.

CreateTCPEndpoint()

Purpose Create the best endpoint object for TCP/IP connections given the available system software.

Access Static, Public

Prototype
```
LTCPEndpoint* CreateTCPEndpoint(UInt32
inReceiveBufferSize)
```

Parameters The parameter for these constructors is:

`UInt32`	`inReceive-BufferSize`	The pointer to the thread.

Return A pointer to the LTCPEndpoint.

CreateUDPEndpoint()

Purpose Create the best endpoint object for TCP/IP connections given the available system software.

Access Static, Public

Prototype
```
LUDPEndpoint* CreateUDPEndpoint();
```

Parameters None

Return A pointer to the LUDPEndpoint.

HasMacTCP()

Purpose Indicates whether MacTCP is present.

Access Static, Public

Prototype `Boolean HasMacTCP();`

Parameters None

Return Return true if MacTCP is present.

HasOpenTransport()

Purpose Indicates whether Open Transport is present.

Access Static, Public

Prototype `Boolean HasOpenTransport();`

Parameters None

Return Return true if OpenTransport is present.

HasTCP()

Purpose Determines whether MacTCP or OpenTransport are installed.

Access Static, Public

Prototype `Boolean HasTCP();`

Parameters None

Return Returns true if either MacTCP or Open Transport are installed.

UOpenTptSupport

Overview UOpenTptSupport is a PowerPlant class that is used for helping other Open Transport classes.

Methods The methods in this class are:

GetOTGestalt()
HasOpenTransport()
HasOpenTransportTCP()
OTAddressToPPAddress()
StartOpenTransport()

Data Members The data members in this class are:

sOTGestaltTested
sCloseOpenTptTask
sOTGestaltResult

Source files (Networking Classes)

`UOpenTptSupport.h`

`UOpenTptSupport.cp`

GetOTGestalt()

Purpose This method is internal only, and is used to perform Open Transport-specific gestalt activity.

Access Static, Private

Prototype

```
static void GetOTGestalt();
```

Parameters None

Return None

HasOpenTransport()

Purpose This method determines whether the machine has Open Transport installed.

Access Static, Public

Prototype
```
static Boolean HasOpenTransport();
```

Parameters None

Return Returns true if Open Transport is installed on the machine.

HasOpenTransportTCP()

Purpose Checks for certain Open Transport gestalt parameters.

Access Static, Public

Prototype
```
static Boolean HasOpenTransportTCP();
```

Parameters None

Return Returns true if certain gestalt parameters for Open Transport are set.

OTAddressToPPAddress()

Purpose Convert an Open Transport address to PowerPlant LInternetAddress.

Access Static, Public

Prototype
```
static LInternetAddress* OTAddressToPPAddress(
     OTAddress* inAddress );
```

Parameters This method has the following parameter:

`OTAddress*`	`inAddress`	The address to convert.

Return An LInternetAddress.

StartOpenTransport()

Purpose Initialize OpenTransport.

Access Static, Public

Prototype
```
static void StartOpenTransport();
```

Parameters None

Return None

sOTGestaltTested

Purpose Whether gestalt has been queried.

Access Static, Private

Prototype
```
Boolean sOTGestaltTested;
```

sCloseOpenTptTask

Purpose Whether to close the Open Transport task or not.

Access Static, Private

Prototype LOpenTpt_CloseOpenTpt* sCloseOpenTptTask;

sOTGestaltResult

Purpose The result of querying `Gestalt()`.

Access Static, Private

Prototype Int32 sOTGestaltResult;

UTextDrawing

Overview UTextDrawing is a PowerPlant class that is used for drawing text in LTextButton and LCaption objects.

Methods There is only one method in this class.

DrawWithJustification()

Data Members There are no data members in this class.

Operation The method in this class provides the same functionality as the MacOS `TextBox()` routine, but does not erase the box before drawing. This improves performance. This means that you must erase the text before drawing if you change it dynamically.

Source files (Utility Classes)

`UDrawingUtils.h`

`UDrawingUtils.cp`

Ancestors None

See also LAttachment

LCaption

LTextButton

DrawWithJustification()

Purpose This method draws text in LTextButton and LCaption objects.

Access Static, Public

Prototype

```
static void DrawWithJustification( Ptr inText,
     Int32 inLength,
     const Rect &inRect,
```

```
    Int16 inJustification,
    Boolean inFirstLeading = true)
```

Parameters The parameters for this method are:

Ptr	inText	A pointer to the text to draw.
Int32	inLength	The length of the text string.
const Rect&	inRect	The `Rect` to draw into.
Int16	inJustification	The justification to use when drawing the text.
Boolean	inFirstLeading	If `true` (the default), then leading will be added above the first line drawn.

Return None

CodeWarrior ®
PowerPlant Reference

Credits

writing lead:	BitHead
other writers:	Many others, of note, Steve Chernicoff
engineering:	Greg Dow, John Daub
frontline warriors:	Jim Trudeau, L. Frank Turovich

Guide to CodeWarrior Documentation

CodeWarrior documentation is modular, like the underlying tools. There are manuals for the core tools, languages, libraries, and targets. The exact documentation provided with any CodeWarrior product is tailored to the tools included with the product. Your product will not have every manual listed here. However, you will probably have additional manuals (not listed here) for utilities or other software specific to your product.

Core Documentation	
IDE User Guide	How to use the CodeWarrior IDE
Debugger User Guide	How to use the CodeWarrior debugger
CodeWarrior Core Tutorials	Step-by-step introduction to IDE components
Language/Compiler Documentation	
C Compilers Reference	Information on the C and C++ compilers
Pascal Compilers Reference	Information on the Pascal and Object Pascal compilers
Error Reference	Comprehensive list of compiler/linker error messages, with many fixes
Pascal Language Reference	The Metrowerks implementation of ANS Pascal
Assembler Guide	Stand-alone assembler manual
Command-Line Tools Reference	Command-line options for Mac OS and Be compilers
Plugin API Manual	The CodeWarrior plugin compiler/linker API
Library Documentation	
MSL C Reference	Function reference for the Metrowerks standard C library
MSL C++ Reference	Function reference for the Metrowerks standard C++ library
Pascal Library Reference	Function reference for the Metrowerks ANS Pascal library
The PowerPlant Book	Guide to the Metrowerks application framework for Mac OS
PowerPlant Advanced Topics	Advanced topics in PowerPlant programming for Mac OS
MFC Reference	Reference for the Microsoft Foundation Classes for Win32
Win32 SDK Reference	Microsoft's Reference for the Win32 API
Targeting Manuals	
Targeting BeOS	How to use CodeWarrior to program for BeOS
Targeting the Java VM	How to use CodeWarrior to program for the Java virtual machine
Targeting Mac OS	How to use CodeWarrior to program for Mac OS
Targeting MIPS	How to use CodeWarrior to program for MIPS embedded processors
Targeting Palm OS	How to use CodeWarrior to program for PalmPilot
Targeting PlayStation OS	How to use CodeWarrior to program for the PlayStation game console
Targeting PowerPC Embedded Systems	How to use CodeWarrior to program for PPC embedded processors
Targeting Win32	How to use CodeWarrior to program for Windows 95/NT